

---

# **STC15F204EA series MCU**

# **STC15L204EA series MCU**

## **Data Sheet**

# CONTENTS

|                                                                    |    |
|--------------------------------------------------------------------|----|
| Chapter 1 Introduction .....                                       | 5  |
| 1.1 Features .....                                                 | 5  |
| 1.2 Block diagram .....                                            | 6  |
| 1.3 PINS Definition.....                                           | 7  |
| 1.3.1 STC15F204EA series Pin Definition.....                       | 7  |
| 1.3.2 STC15F101E series Pin Definition .....                       | 8  |
| 1.3.3 STC15S204EA series Pin Definition.....                       | 9  |
| 1.4 STC15F204EA series Typical Application Circuit (for ISP) ..... | 10 |
| 1.5 PINS Descriptions of STC15F204EA series.....                   | 11 |
| 1.6 Package Drawings.....                                          | 13 |
| 1.7 STC15Fxx series MCU naming rules .....                         | 19 |
| 1.7.1 STC15F204EA series MCU naming rules.....                     | 19 |
| 1.7.2 STC15F101E series MCU naming rules .....                     | 20 |
| Chapter 2 Clock, Power Management, Reset .....                     | 21 |
| 2.1 Clock .....                                                    | 21 |
| 2.2 Power Management.....                                          | 22 |
| 2.2.1 Idle Mode.....                                               | 22 |
| 2.2.2 Slow Down Mode .....                                         | 23 |
| 2.2.3 Power Down (PD) Mode (Stop Mode).....                        | 24 |
| 2.3 Reset.....                                                     | 30 |
| 2.3.1 Reset pin .....                                              | 30 |
| 2.3.2 Software RESET .....                                         | 30 |
| 2.3.3 Power-On Reset (POR) .....                                   | 30 |
| 2.3.4 Watch-Dog-Timer .....                                        | 31 |
| 2.3.5 MAX810 power-on-reset delay .....                            | 35 |
| 2.3.6 Low Voltage Detection .....                                  | 35 |
| Chapter 3 Memory Organization .....                                | 37 |
| 3.1 Program Memory .....                                           | 37 |
| 3.2 SRAM .....                                                     | 38 |
| Chapter 4 Configurable I/O Ports .....                             | 40 |
| 4.1 I/O Port Configurations.....                                   | 40 |
| 4.1.1 Quasi-bidirectional I/O .....                                | 40 |
| 4.1.2 Push-pull Output.....                                        | 41 |
| 4.1.3 Input-only Mode .....                                        | 41 |
| 4.1.4 Open-drain Output.....                                       | 41 |

|                                                                            |     |
|----------------------------------------------------------------------------|-----|
| 4.2 I/O Port Registers .....                                               | 42  |
| 4.3 I/O port application notes.....                                        | 44  |
| 4.4 I/O port application .....                                             | 44  |
| 4.4.1 Typical transistor control circuit .....                             | 44  |
| 4.4.2 Typical diode control circuit .....                                  | 44  |
| 4.4.3 3V/5V hybrid system .....                                            | 45  |
| 4.4.4 How to make I/O port low after MCU reset.....                        | 45  |
| 4.4.5 I/O drive LED application circuit .....                              | 46  |
| 4.4.6 I/O immediately drive LCD application circuit.....                   | 47  |
| 4.4.7 Using A/D Conversion to scan key application circuit.....            | 48  |
| Chapter 5 Instruction System.....                                          | 49  |
| 5.1 Special Function Registers.....                                        | 49  |
| 5.2 Notes on Compatibility to Standard 80C51 MCU .....                     | 53  |
| 5.3 Addressing Modes .....                                                 | 54  |
| 5.4 Instruction Set Summary .....                                          | 55  |
| 5.5 Instruction Definitions for Standard 8051 MCU .....                    | 60  |
| Chapter 6 Interrupts .....                                                 | 97  |
| 6.1 Interrupt Structure .....                                              | 98  |
| 6.2 Interrupt Register .....                                               | 100 |
| 6.3 Interrupt Priorities .....                                             | 103 |
| 6.4 How Interrupts Are Handled .....                                       | 104 |
| 6.5 External Interrupts.....                                               | 106 |
| Chapter 7 Timer/Counter 0 and 1.....                                       | 116 |
| 7.1 Timer/Counter 0 Mode of Operation.....                                 | 119 |
| 7.2 Timer/Counter 1 Mode of Operation.....                                 | 126 |
| 7.3 Generic Programmable Clock Output .....                                | 131 |
| 7.4 Changes of STC15F204EA series Timers compared with standard 8051 ..... | 138 |
| Chapter 8 Simulate Serial Port Program.....                                | 140 |
| 8.1 Programs using Timer 0 to realize Simulate Serial Port .....           | 140 |
| 8.2 Programs using Timer 1 to realize Simulate Serial Port .....           | 149 |

|                                                                 |     |
|-----------------------------------------------------------------|-----|
| Chapter 9 Analog to Digital Converter .....                     | 158 |
| 9.1 A/D Converter Structure .....                               | 158 |
| 9.2 Register for ADC.....                                       | 160 |
| 9.3 Program using interrupts to demonstrate ADC .....           | 162 |
| 9.4 Program using polling to demonstrate ADC .....              | 172 |
| Chapter 10 IAP / EEPROM .....                                   | 183 |
| 10.1 IAP / ISP Control Register .....                           | 183 |
| 10.2 IAP/EEPROM Assembly Language Program Introduction .....    | 186 |
| 10.3 EEPROM Demo Programs written in Assembly Language .....    | 188 |
| 10.4 EEPROM Demo Program written in C Language .....            | 199 |
| Chapter 11 STC15Fxx series programming tools usage.....         | 210 |
| 11.1 In-System-Programming (ISP) principle .....                | 210 |
| 11.2 STC15F204EA series application circuit for ISP.....        | 211 |
| 11.3 PC side application usage.....                             | 212 |
| 11.4 Compiler / Assembler Programmer and Emulator .....         | 214 |
| 11.5 Self-Defined ISP download Demo .....                       | 214 |
| Appendix A: Assembly Language Programming .....                 | 217 |
| Appendix B: 8051 C Programming .....                            | 239 |
| Appendix C: STC15F204EA series Electrical Characteristics ..... | 249 |
| Appendix D: STC15Fxx series to replace standard 8051 Notes...   | 250 |
| Appendix E: STC15F204EA series Selection Table.....             | 252 |

---

# Chapter 1 Introduction

STC15F204EA series 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, STC15F204EA series execute 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-volatile data in Flash memory while the application program is running. the STC15F204EA series has 9 interrupt sources, 10-bit ADC, on-chip high-precision RC oscillator and a one-time enabled Watch-Dog Timer.

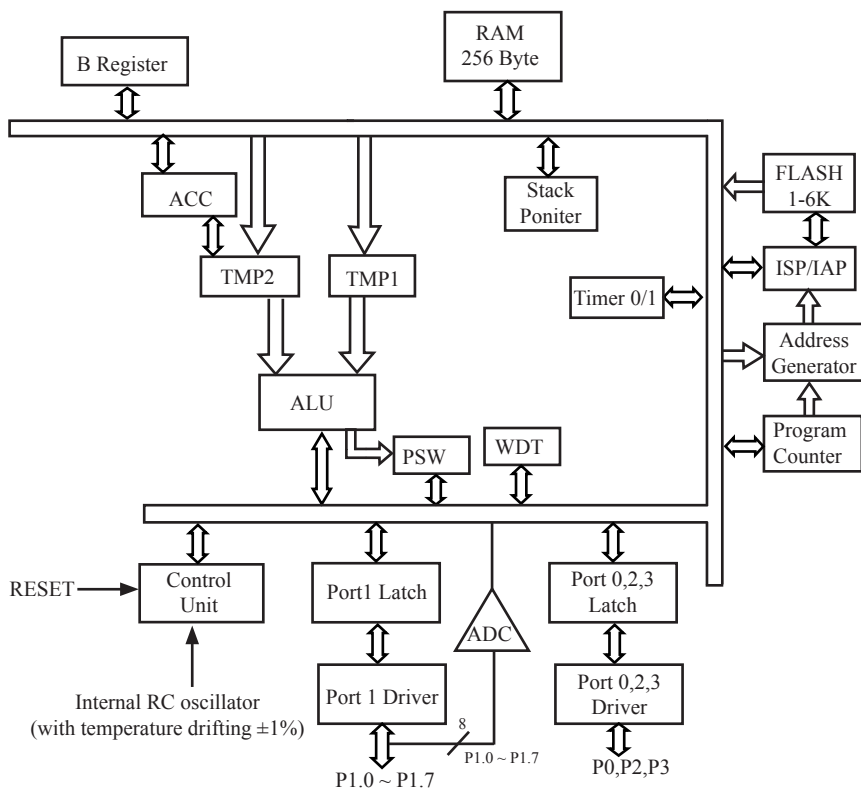
## 1.1 Features

- 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 (STC15L204EA series)
- Operating frequency range: 5MHz ~ 35MHz, is equivalent to standard 8051: 60MHz ~ 420MHz
- A high-precision internal RC oscillator with temperature drifting  $\pm 1\%$  ( $-40^{\circ}\text{C} \sim +85^{\circ}\text{C}$ )
- internal RC oscillator with adjustable frequency to 5.5296MHz/11.0592MHz/22.1184MHz/33.1776MHz
- On-chip 256 bytes RAM and 1K~6K bytes code flash with flexible ISP/IAP capability
- EEPROM function
- Code protection for flash memory access
- Two 16-bit timers/counters — Timer 0 / Timer 1 with mode 0 (16-bit auto-reload mode), mode 1 (16-bit timer mode) and mode 2 (8-bit auto-reload mode)
- simulate UART can be realized by P3.0, P3.1 and Timers
- 8-channel, 10-bit ADC associated interrupt, speed up to 300 thousands times every second
- 9 interrupt sources
- One 15 bits Watch-Dog-Timer with 8-bit pre-scalar (one-time-enabled)
- Three power management modes: idle mode, slow down mode and power-down mode  
Power down mode can be woken-up by external INTx pin (INT0/P3.2, INT1/P3.3,  $\overline{\text{INT2}}$ ,  $\overline{\text{INT3}}$ ,  $\overline{\text{INT4}}$ )
- Excellent noise immunity, very low power consumption
- 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.
- 26 configurable I/O ports are available and default to quasi-bidirectional after reset. All ports may be independently configured to one of four modes : quasi-bidirectional, push-pull output, input-only or open-drain output. The drive capability of each port is up to 20 mA. But recommend the whole chip's should be less than 90 mA.
- Package type: SOP-28,SKDIP-28

---

## 1.2 Block diagram

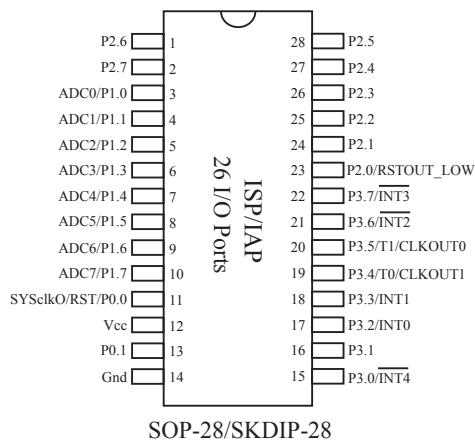
The CPU kernel of STC15F204EA series is fully compatible to the standard 8051 microcontroller, maintains all instruction mnemonics and binary compatibility. With some great architecture enhancements, STC15F204EA series execute the fastest instructions per clock cycle. Improvement of individual programs depends on the actual instructions used.



STC15F204EA series Block Diagram

## 1.3 PINS Definition

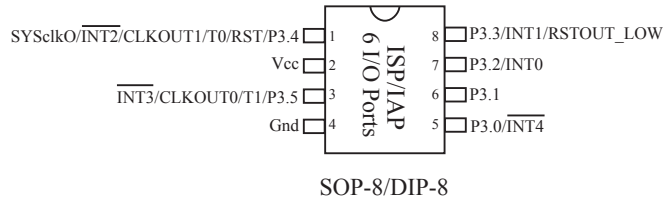
### 1.3.1 STC15F204EA series Pin Definition



STC15F204EA series Selection Table

| Type<br>1T 8051 MCU | Operating<br>voltage<br>(V) | Flash<br>(B) | SRAM<br>(B) | TIMER | A/D    | WD<br>T | EEP<br>ROM<br>(B) | Internal<br>low<br>voltage<br>interrupt | Internal<br>Reset<br>threshold<br>voltage<br>can be<br>configured | External<br>interrupts<br>which can<br>wake up<br>power down<br>mode | Special<br>timer for<br>waking<br>power<br>down<br>mode | Package of 28-pin<br>(26 I/O ports)<br>Price (RMB ¥) |          |
|---------------------|-----------------------------|--------------|-------------|-------|--------|---------|-------------------|-----------------------------------------|-------------------------------------------------------------------|----------------------------------------------------------------------|---------------------------------------------------------|------------------------------------------------------|----------|
|                     |                             |              |             |       |        |         |                   |                                         |                                                                   |                                                                      |                                                         | SOP-28                                               | SKDIP-28 |
| STC15F201A          | 5.5~3.8                     | 1K           | 256         | 2     | 10-bit | Y       | -                 | Y                                       | Y                                                                 | 5                                                                    | N                                                       |                                                      |          |
| STC15F201EA         | 5.5~3.8                     | 1K           | 256         | 2     | 10-bit | Y       | 2K                | Y                                       | Y                                                                 | 5                                                                    | N                                                       | ¥                                                    | ¥        |
| STC15F202A          | 5.5~3.8                     | 2K           | 256         | 2     | 10-bit | Y       | -                 | Y                                       | Y                                                                 | 5                                                                    | N                                                       |                                                      |          |
| STC15F202EA         | 5.5~3.8                     | 2K           | 256         | 2     | 10-bit | Y       | 2K                | Y                                       | Y                                                                 | 5                                                                    | N                                                       | ¥                                                    | ¥        |
| STC15F203A          | 5.5~3.8                     | 3K           | 256         | 2     | 10-bit | Y       | -                 | Y                                       | Y                                                                 | 5                                                                    | N                                                       |                                                      |          |
| STC15F203EA         | 5.5~3.8                     | 3K           | 256         | 2     | 10-bit | Y       | 2K                | Y                                       | Y                                                                 | 5                                                                    | N                                                       | ¥                                                    | ¥        |
| STC15F204A          | 5.5~3.8                     | 4K           | 256         | 2     | 10-bit | Y       | -                 | Y                                       | Y                                                                 | 5                                                                    | N                                                       |                                                      |          |
| STC15F204EA         | 5.5~3.8                     | 4K           | 256         | 2     | 10-bit | Y       | 1K                | Y                                       | Y                                                                 | 5                                                                    | N                                                       | ¥                                                    | ¥        |
| STC15F205A          | 5.5~3.8                     | 5K           | 256         | 2     | 10-bit | Y       | -                 | Y                                       | Y                                                                 | 5                                                                    | N                                                       |                                                      |          |
| STC15F205EA         | 5.5~3.8                     | 5K           | 256         | 2     | 10-bit | Y       | 1K                | Y                                       | Y                                                                 | 5                                                                    | N                                                       | ¥                                                    | ¥        |
| IAP15F206A          | 5.5~3.8                     | 6K           | 256         | 2     | 10-bit | Y       | IAP               | Y                                       | Y                                                                 | 5                                                                    | N                                                       |                                                      |          |
| STC15L201A          | 3.6~2.4                     | 1K           | 256         | 2     | 10-bit | Y       | -                 | Y                                       | Y                                                                 | 5                                                                    | N                                                       |                                                      |          |
| STC15L201EA         | 3.6~2.4                     | 1K           | 256         | 2     | 10-bit | Y       | 2K                | Y                                       | Y                                                                 | 5                                                                    | N                                                       | ¥                                                    | ¥        |
| STC15L202A          | 3.6~2.4                     | 2K           | 256         | 2     | 10-bit | Y       | -                 | Y                                       | Y                                                                 | 5                                                                    | N                                                       |                                                      |          |
| STC15L202EA         | 3.6~2.4                     | 2K           | 256         | 2     | 10-bit | Y       | 2K                | Y                                       | Y                                                                 | 5                                                                    | N                                                       | ¥                                                    | ¥        |
| STC15L203A          | 3.6~2.4                     | 3K           | 256         | 2     | 10-bit | Y       | -                 | Y                                       | Y                                                                 | 5                                                                    | N                                                       |                                                      |          |
| STC15L203EA         | 3.6~2.4                     | 3K           | 256         | 2     | 10-bit | Y       | 2K                | Y                                       | Y                                                                 | 5                                                                    | N                                                       | ¥                                                    | ¥        |
| STC15L204A          | 3.6~2.4                     | 4K           | 256         | 2     | 10-bit | Y       | -                 | Y                                       | Y                                                                 | 5                                                                    | N                                                       |                                                      |          |
| STC15L204EA         | 3.6~2.4                     | 4K           | 256         | 2     | 10-bit | Y       | 1K                | Y                                       | Y                                                                 | 5                                                                    | N                                                       | ¥                                                    | ¥        |
| STC15L205A          | 3.6~2.4                     | 5K           | 256         | 2     | 10-bit | Y       | -                 | Y                                       | Y                                                                 | 5                                                                    | N                                                       |                                                      |          |
| STC15L205EA         | 3.6~2.4                     | 5K           | 256         | 2     | 10-bit | Y       | 1K                | Y                                       | Y                                                                 | 5                                                                    | N                                                       | ¥                                                    | ¥        |
| IAP15L206A          | 3.6~2.4                     | 6K           | 256         | 2     | 10-bit | Y       | IAP               | Y                                       | Y                                                                 | 5                                                                    | N                                                       |                                                      |          |

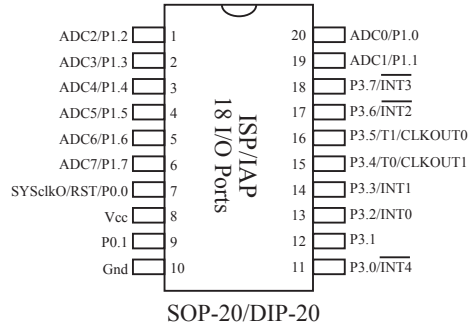
### 1.3.2 STC15F101E series Pin Definition



STC15F101EA series Selection Table

| Type<br>1T 8051 MCU | Operating<br>voltage<br>(V) | Flash<br>(B) | S<br>R<br>A<br>M<br>(B) | Timer | A/D | W<br>D<br>T | EEP<br>ROM<br>(B) | Internal<br>low<br>voltage<br>interrupt | Internal<br>Reset<br>threshold<br>voltage<br>can be<br>configured | External<br>interrupts<br>which<br>can wake<br>up power<br>down mode | Special<br>timer for<br>waking<br>power<br>down<br>mode | Package of 8-pin<br>(6 I/O ports)<br>Price (RMB ¥) |       |
|---------------------|-----------------------------|--------------|-------------------------|-------|-----|-------------|-------------------|-----------------------------------------|-------------------------------------------------------------------|----------------------------------------------------------------------|---------------------------------------------------------|----------------------------------------------------|-------|
|                     |                             |              |                         |       |     |             |                   |                                         |                                                                   |                                                                      |                                                         | SOP-8                                              | DIP-8 |
| STC15F100           | 5.5~3.8                     | 512          | 128                     | 2     | -   | Y           | -                 | Y                                       | Y                                                                 | 5                                                                    | N                                                       | ¥                                                  | ¥     |
| STC15F101           | 5.5~3.8                     | 1K           | 128                     | 2     | -   | Y           | -                 | Y                                       | Y                                                                 | 5                                                                    | N                                                       | ¥                                                  | ¥     |
| STC15F101E          | 5.5~3.8                     | 1K           | 128                     | 2     | -   | Y           | 2K                | Y                                       | Y                                                                 | 5                                                                    | N                                                       | ¥                                                  | ¥     |
| STC15F102           | 5.5~3.8                     | 2K           | 128                     | 2     | -   | Y           | -                 | Y                                       | Y                                                                 | 5                                                                    | N                                                       | ¥                                                  | ¥     |
| STC15F102E          | 5.5~3.8                     | 2K           | 128                     | 2     | -   | Y           | 2K                | Y                                       | Y                                                                 | 5                                                                    | N                                                       | ¥                                                  | ¥     |
| STC15F103           | 5.5~3.8                     | 3K           | 128                     | 2     | -   | Y           | -                 | Y                                       | Y                                                                 | 5                                                                    | N                                                       | ¥                                                  | ¥     |
| STC15F103E          | 5.5~3.8                     | 3K           | 128                     | 2     | -   | Y           | 2K                | Y                                       | Y                                                                 | 5                                                                    | N                                                       | ¥                                                  | ¥     |
| STC15F104           | 5.5~3.8                     | 4K           | 128                     | 2     | -   | Y           | -                 | Y                                       | Y                                                                 | 5                                                                    | N                                                       | ¥                                                  | ¥     |
| STC15F104E          | 5.5~3.8                     | 4K           | 128                     | 2     | -   | Y           | 1K                | Y                                       | Y                                                                 | 5                                                                    | N                                                       | ¥                                                  | ¥     |
| STC15F105           | 5.5~3.8                     | 5K           | 128                     | 2     | -   | Y           | -                 | Y                                       | Y                                                                 | 5                                                                    | N                                                       |                                                    |       |
| STC15F105E          | 5.5~3.8                     | 5K           | 128                     | 2     | -   | Y           | 1K                | Y                                       | Y                                                                 | 5                                                                    | N                                                       |                                                    |       |
| IAP15F106           | 5.5~3.8                     | 6K           | 128                     | 2     | -   | Y           | IAP               | Y                                       | Y                                                                 | 5                                                                    | N                                                       |                                                    |       |
| Type<br>1T 8051 MCU | Operating<br>voltage<br>(V) | Flash<br>(B) | S<br>R<br>A<br>M<br>(B) | Timer | A/D | W<br>D<br>T | EEP<br>ROM<br>(B) | Internal<br>low<br>voltage<br>interrupt | Internal<br>Reset<br>threshold<br>voltage<br>can be<br>configured | External<br>interrupts<br>which<br>can wake<br>up power<br>down mode | Special<br>timer for<br>waking<br>power<br>down<br>mode | Package of 8-pin<br>(6 I/O ports)<br>Price (RMB ¥) |       |
|                     |                             |              |                         |       |     |             |                   |                                         |                                                                   |                                                                      |                                                         | SOP-8                                              | DIP-8 |
| STC15L100           | 3.6~2.4                     | 512          | 128                     | 2     | -   | Y           | -                 | Y                                       | Y                                                                 | 5                                                                    | N                                                       | ¥                                                  | ¥     |
| STC15L101           | 3.6~2.4                     | 1K           | 128                     | 2     | -   | Y           | -                 | Y                                       | Y                                                                 | 5                                                                    | N                                                       | ¥                                                  | ¥     |
| STC15L101E          | 3.6~2.4                     | 1K           | 128                     | 2     | -   | Y           | 2K                | Y                                       | Y                                                                 | 5                                                                    | N                                                       | ¥                                                  | ¥     |
| STC15L102           | 3.6~2.4                     | 2K           | 128                     | 2     | -   | Y           | -                 | Y                                       | Y                                                                 | 5                                                                    | N                                                       | ¥                                                  | ¥     |
| STC15L102E          | 3.6~2.4                     | 2K           | 128                     | 2     | -   | Y           | 2K                | Y                                       | Y                                                                 | 5                                                                    | N                                                       | ¥                                                  | ¥     |
| STC15L103           | 3.6~2.4                     | 3K           | 128                     | 2     | -   | Y           | -                 | Y                                       | Y                                                                 | 5                                                                    | N                                                       | ¥                                                  | ¥     |
| STC15L103E          | 3.6~2.4                     | 3K           | 128                     | 2     | -   | Y           | 2K                | Y                                       | Y                                                                 | 5                                                                    | N                                                       | ¥                                                  | ¥     |
| STC15L104           | 3.6~2.4                     | 4K           | 128                     | 2     | -   | Y           | -                 | Y                                       | Y                                                                 | 5                                                                    | N                                                       | ¥                                                  | ¥     |
| STC15L104E          | 3.6~2.4                     | 4K           | 128                     | 2     | -   | Y           | 1K                | Y                                       | Y                                                                 | 5                                                                    | N                                                       | ¥                                                  | ¥     |
| STC15L105           | 3.6~2.4                     | 5K           | 128                     | 2     | -   | Y           | -                 | Y                                       | Y                                                                 | 5                                                                    | N                                                       |                                                    |       |
| STC15L105E          | 3.6~2.4                     | 5K           | 128                     | 2     | -   | Y           | 1K                | Y                                       | Y                                                                 | 5                                                                    | N                                                       |                                                    |       |
| IAP15L106           | 3.6~2.4                     | 6K           | 128                     | 2     | -   | Y           | IAP               | Y                                       | Y                                                                 | 5                                                                    | N                                                       |                                                    |       |

### 1.3.3 STC15S204EA series Pin Definition

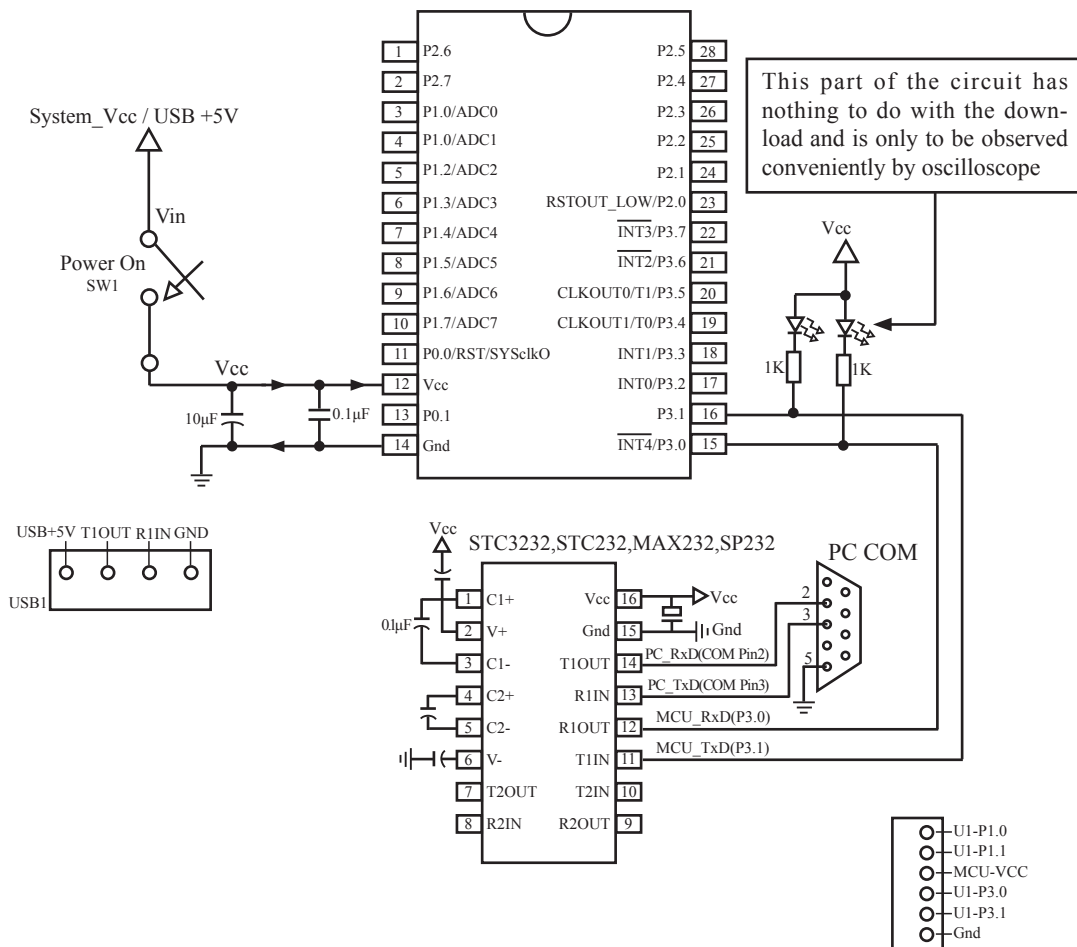


STC15S204EA series is the special version of STC15F204EA series MCU, but it has no sample provided currently.

STC15S204EA series Selection Table

| Type<br>1T 8051 MCU | Operating<br>voltage<br>(V) | Flash<br>(B) | S<br>R<br>A<br>M<br>(B) | T<br>I<br>M<br>E<br>R | A/D    | W<br>D<br>T | EEP<br>ROM<br>(B) | Internal<br>low<br>voltage<br>interrupt | Internal<br>Reset<br>threshold<br>voltage<br>can be<br>configured | External<br>interrupts<br>which<br>can wake<br>up power<br>down<br>mode | Special<br>timer<br>for<br>waking<br>power<br>down<br>mode | Package of 20-pin<br>(18 I/O ports)<br>Price (RMB ¥) |        |
|---------------------|-----------------------------|--------------|-------------------------|-----------------------|--------|-------------|-------------------|-----------------------------------------|-------------------------------------------------------------------|-------------------------------------------------------------------------|------------------------------------------------------------|------------------------------------------------------|--------|
|                     |                             |              |                         |                       |        |             |                   |                                         |                                                                   |                                                                         |                                                            | SOP-20                                               | DIP-20 |
| STC15S201A          | 5.5~3.8                     | 1K           | 256                     | 2                     | 10-bit | Y           | -                 | Y                                       | Y                                                                 | 5                                                                       | N                                                          |                                                      |        |
| STC15S201EA         | 5.5~3.8                     | 1K           | 256                     | 2                     | 10-bit | Y           | 2K                | Y                                       | Y                                                                 | 5                                                                       | N                                                          |                                                      |        |
| STC15S202A          | 5.5~3.8                     | 2K           | 256                     | 2                     | 10-bit | Y           | -                 | Y                                       | Y                                                                 | 5                                                                       | N                                                          |                                                      |        |
| STC15S202EA         | 5.5~3.8                     | 2K           | 256                     | 2                     | 10-bit | Y           | 2K                | Y                                       | Y                                                                 | 5                                                                       | N                                                          |                                                      |        |
| STC15S203A          | 5.5~3.8                     | 3K           | 256                     | 2                     | 10-bit | Y           | -                 | Y                                       | Y                                                                 | 5                                                                       | N                                                          |                                                      |        |
| STC15S203EA         | 5.5~3.8                     | 3K           | 256                     | 2                     | 10-bit | Y           | 2K                | Y                                       | Y                                                                 | 5                                                                       | N                                                          |                                                      |        |
| STC15S204A          | 5.5~3.8                     | 4K           | 256                     | 2                     | 10-bit | Y           | -                 | Y                                       | Y                                                                 | 5                                                                       | N                                                          |                                                      |        |
| STC15S204EA         | 5.5~3.8                     | 4K           | 256                     | 2                     | 10-bit | Y           | 1K                | Y                                       | Y                                                                 | 5                                                                       | N                                                          |                                                      |        |
| STC15S205A          | 5.5~3.8                     | 5K           | 256                     | 2                     | 10-bit | Y           | -                 | Y                                       | Y                                                                 | 5                                                                       | N                                                          |                                                      |        |
| STC15S205EA         | 5.5~3.8                     | 5K           | 256                     | 2                     | 10-bit | Y           | 1K                | Y                                       | Y                                                                 | 5                                                                       | N                                                          |                                                      |        |
| IAP15S206A          | 5.5~3.8                     | 6K           | 256                     | 2                     | 10-bit | Y           | IAP               | Y                                       | Y                                                                 | 5                                                                       | N                                                          |                                                      |        |
| STC15V201A          | 3.6~2.4                     | 1K           | 256                     | 2                     | 10-bit | Y           | -                 | Y                                       | Y                                                                 | 5                                                                       | N                                                          |                                                      |        |
| STC15V201EA         | 3.6~2.4                     | 1K           | 256                     | 2                     | 10-bit | Y           | 2K                | Y                                       | Y                                                                 | 5                                                                       | N                                                          |                                                      |        |
| STC15V202A          | 3.6~2.4                     | 2K           | 256                     | 2                     | 10-bit | Y           | -                 | Y                                       | Y                                                                 | 5                                                                       | N                                                          |                                                      |        |
| STC15V202EA         | 3.6~2.4                     | 2K           | 256                     | 2                     | 10-bit | Y           | 2K                | Y                                       | Y                                                                 | 5                                                                       | N                                                          |                                                      |        |
| STC15V203A          | 3.6~2.4                     | 3K           | 256                     | 2                     | 10-bit | Y           | -                 | Y                                       | Y                                                                 | 5                                                                       | N                                                          |                                                      |        |
| STC15V203EA         | 3.6~2.4                     | 3K           | 256                     | 2                     | 10-bit | Y           | 2K                | Y                                       | Y                                                                 | 5                                                                       | N                                                          |                                                      |        |
| STC15V204A          | 3.6~2.4                     | 4K           | 256                     | 2                     | 10-bit | Y           | -                 | Y                                       | Y                                                                 | 5                                                                       | N                                                          |                                                      |        |
| STC15V204EA         | 3.6~2.4                     | 4K           | 256                     | 2                     | 10-bit | Y           | 1K                | Y                                       | Y                                                                 | 5                                                                       | N                                                          |                                                      |        |
| STC15V205A          | 3.6~2.4                     | 5K           | 256                     | 2                     | 10-bit | Y           | -                 | Y                                       | Y                                                                 | 5                                                                       | N                                                          |                                                      |        |
| STC15V205EA         | 3.6~2.4                     | 5K           | 256                     | 2                     | 10-bit | Y           | 1K                | Y                                       | Y                                                                 | 5                                                                       | N                                                          |                                                      |        |
| IAP15V206A          | 3.6~2.4                     | 6K           | 256                     | 2                     | 10-bit | Y           | IAP               | Y                                       | Y                                                                 | 5                                                                       | N                                                          |                                                      |        |

## 1.4 STC15F204EA series Typical Application Circuit (for ISP)



On-chip high-reliability Reset, No need external Reset circuit

Internal high-precision RC oscillator with temperature drifting  $\pm 1\%$  ( $-40^{\circ}\text{C} \sim +80^{\circ}\text{C}$ ), No need expensive external crystal oscillator.

## 1.5 PINS Descriptions of STC15F204EA series

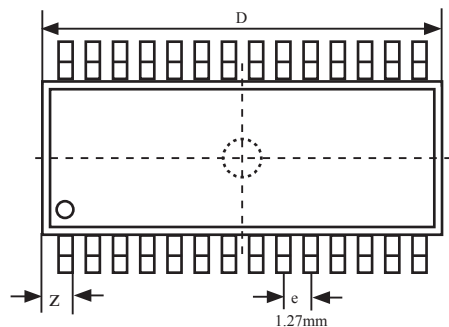
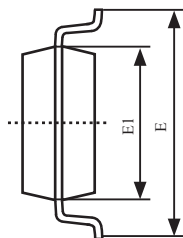
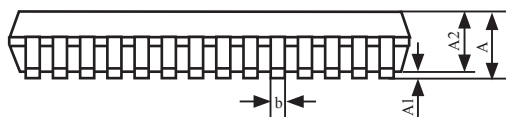
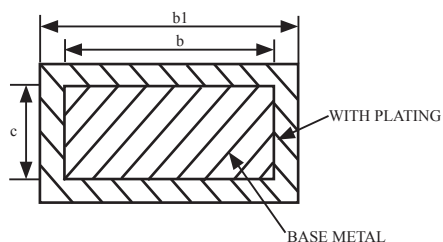
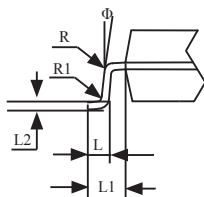
| MNEMONIC                       | Pin number | DESCRIPTION              |                                                                                                                                                                             |
|--------------------------------|------------|--------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| P0.0/RST/SYScLK                | 11         | P0.0                     | Standard PORT0[0]                                                                                                                                                           |
|                                |            | RST                      | Reset pin;                                                                                                                                                                  |
|                                |            | SYScLK                   | 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/<br>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 input                                                                                    |
| 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.<br>The interrupting acts in Negative-Edge only, and with Lease priority, and it can wake up the STC15F204EA series 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.<br>The interrupt acting can be configured to Negative-Edge-Active or On-Change-Active(Negative-Edge-Active and Positive-Edge-Active).<br>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.<br>Also INT0 can wake up the STC15F204EA series from power-down mode. |
| P3.3/INT1                      | 18         | P3.3                     | Standard PORT3[3]                                                                                                                                                                                                                                                                                                                                                                                                                 |
|                                |            | INT1                     | One of external Interrupt sources.<br>The interrupt acting can be configured to Negative-Edge-Active or On-Change-Active(Negative-Edge-Active and Positive-Edge-Active).<br>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.<br>Also INT1 can wake up the STC15F204EA series 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<br>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<br>Set INT_CLKO[0](T0CLKO)=1 to act it.                                                                                                                                                                                                                                                                                                                       |
| P3.6/ $\overline{\text{INT2}}$ | 21         | P3.6                     | Standard PORT3[6]                                                                                                                                                                                                                                                                                                                                                                                                                 |
|                                |            | $\overline{\text{INT2}}$ | One of external Interrupt sources.<br>The interrupting acts in Negative-Edge only, and with Lease priority, and it can wake up the STC15F204EA series from power-down mode.                                                                                                                                                                                                                                                       |
| P3.7/ $\overline{\text{INT3}}$ | 22         | P3.7                     | Standard PORT3[7]                                                                                                                                                                                                                                                                                                                                                                                                                 |
|                                |            | $\overline{\text{INT3}}$ | One of external Interrupt sources.<br>The interrupting acts in Negative-Edge only, and with Lease priority, and it can wake up the STC15F204EA series from power-down mode.                                                                                                                                                                                                                                                       |
| Vcc                            | 12         | Power                    |                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| Gnd                            | 14         | Ground                   |                                                                                                                                                                                                                                                                                                                                                                                                                                   |

## 1.6 Package Drawings

### 28-Pin Small Outline Package (SOP-28)

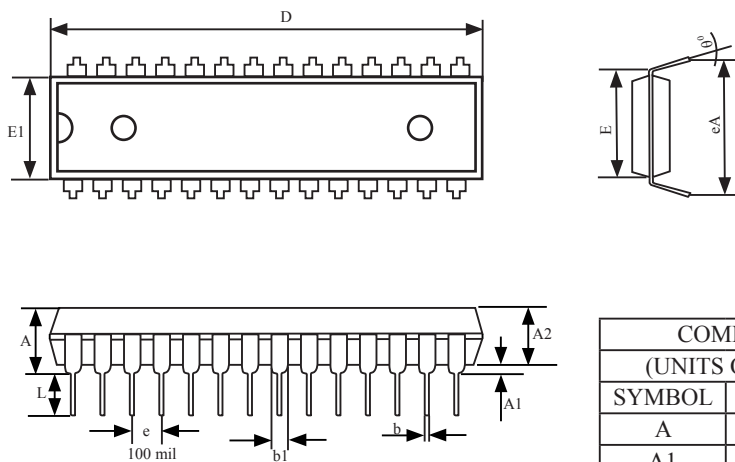
Dimensions in Millimeters



| COMMON DIMENSIONS                    |                |        |                 |
|--------------------------------------|----------------|--------|-----------------|
| (UNITS OF MEASURE = MILLIMETER / mm) |                |        |                 |
| 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 <sup>0</sup> | -      | 10 <sup>0</sup> |
| z                                    | -              | 0.745  | -               |

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

Dimensions in Inches and Millimeters

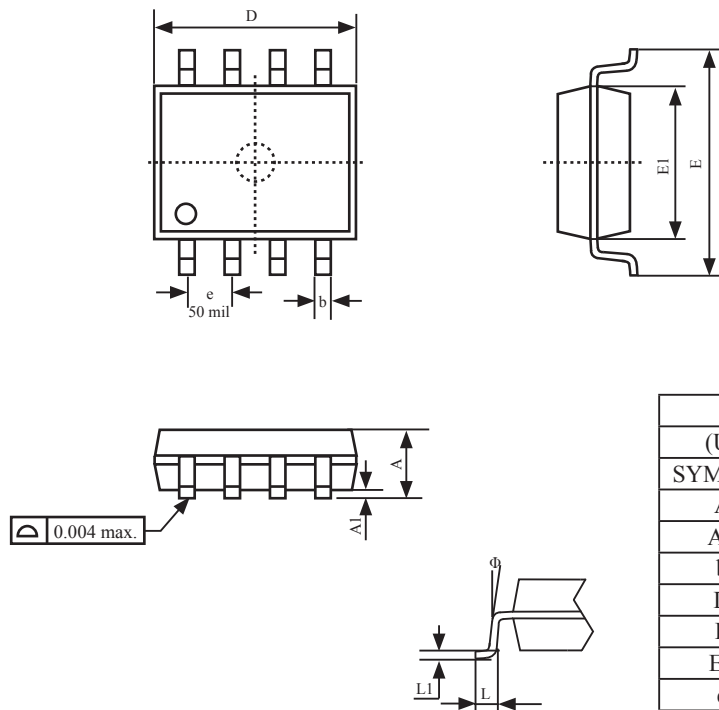


| COMMON DIMENSIONS         |       |       |       |
|---------------------------|-------|-------|-------|
| (UNITS OF MEASURE = INCH) |       |       |       |
| SYMBOL                    | MIN   | NOM   | MAX   |
| A                         | -     | -     | 0.210 |
| A1                        | 0.015 | -     | -     |
| 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 |
| $\theta^0$                | 0     | 7     | 15    |
| eA                        | 0.330 | 0.350 | 0.370 |

UNIT: INCH, 1 inch = 1000 mil

## 8-PIN SMALL OUTLINE PACKAGE (SOP-8)

Dimensions in Inches

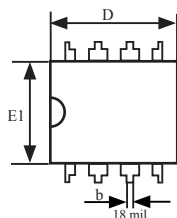
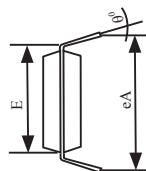
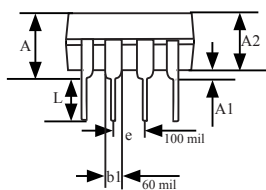


| COMMON DIMENSIONS         |       |       |       |
|---------------------------|-------|-------|-------|
| (UNITS OF MEASURE = INCH) |       |       |       |
| SYMBOL                    | MIN   | NOM   | MAX   |
| A                         | 0.053 | -     | 0.069 |
| A1                        | 0.004 | -     | 0.010 |
| b                         | -     | 0.016 | -     |
| D                         | 0.189 | -     | 0.196 |
| E                         | 0.228 | -     | 0.244 |
| E1                        | 0.150 | -     | 0.157 |
| e                         | 0.050 |       |       |
| L                         | 0.016 | -     | 0.050 |
| L1                        | 0.008 |       |       |
| Φ                         | 0°    | -     | 8°    |

UNIT: INCH, 1 inch = 1000 mil

## 8-Pin Plastic Dual Inline Package (DIP-8)

Dimensions in Inches

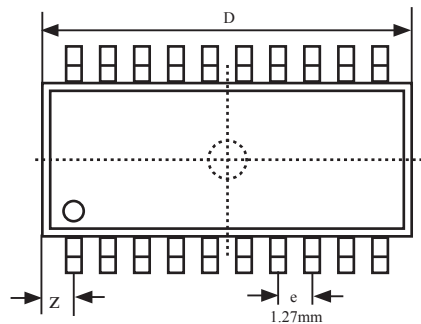
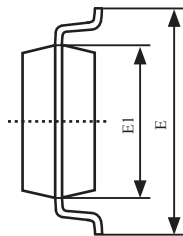
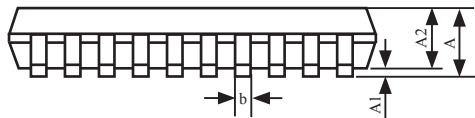
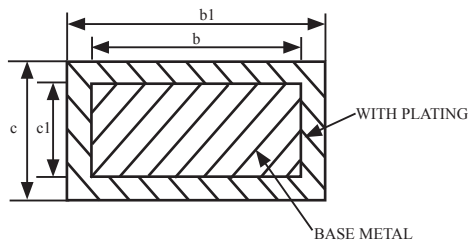
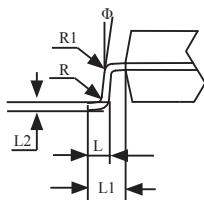


| COMMON DIMENSIONS         |       |       |       |
|---------------------------|-------|-------|-------|
| (UNITS OF MEASURE = INCH) |       |       |       |
| SYMBOL                    | MIN   | NOM   | MAX   |
| A                         | -     | -     | 0.210 |
| A1                        | 0.015 | -     | -     |
| A2                        | 0.125 | 0.130 | 0.135 |
| b                         | -     | 0.018 | -     |
| b1                        | -     | 0.060 | -     |
| D                         | 0.355 | 0.365 | 0.400 |
| E                         | -     | 0.300 | -     |
| E1                        | 0.245 | 0.250 | 0.255 |
| e                         | -     | 0.100 | -     |
| L                         | 0.115 | 0.130 | 0.150 |
| θ°                        | 0     | 7     | 15    |
| eA                        | 0.335 | 0.355 | 0.375 |

UNIT: INCH, 1 inch = 1000 mil

20-Pin Small Outline Package (SOP-20) (for STC15S/V204EA series)

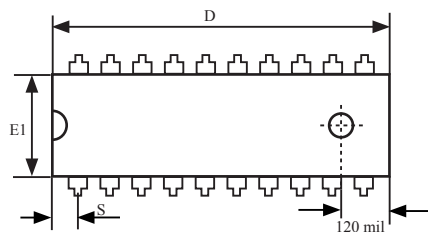
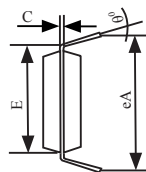
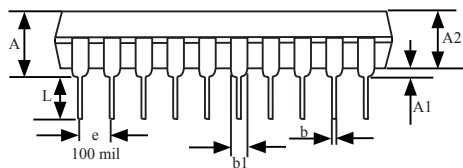
Dimensions in Inches and (Millimeters)



| COMMON DIMENSIONS               |                |        |                 |
|---------------------------------|----------------|--------|-----------------|
| (UNITS OF MEASURE = MILLIMETER) |                |        |                 |
| 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           |
| b1                              | 0.366          | 0.426  | 0.486           |
| b                               | 0.356          | 0.406  | 0.456           |
| c                               | 0.234          | -      | 0.274           |
| c1                              | 0.224          | 0.254  | 0.274           |
| 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                               | -              | 0.300  | -               |
| R1                              | -              | 0.200  | -               |
| Φ                               | 0 <sup>0</sup> | -      | 10 <sup>0</sup> |
| z                               | -              | 0.660  | -               |

## 20-Pin Plastic Dual Inline Package (DIP-20) (for STC15S/V204EA series)

Dimensions in Inches



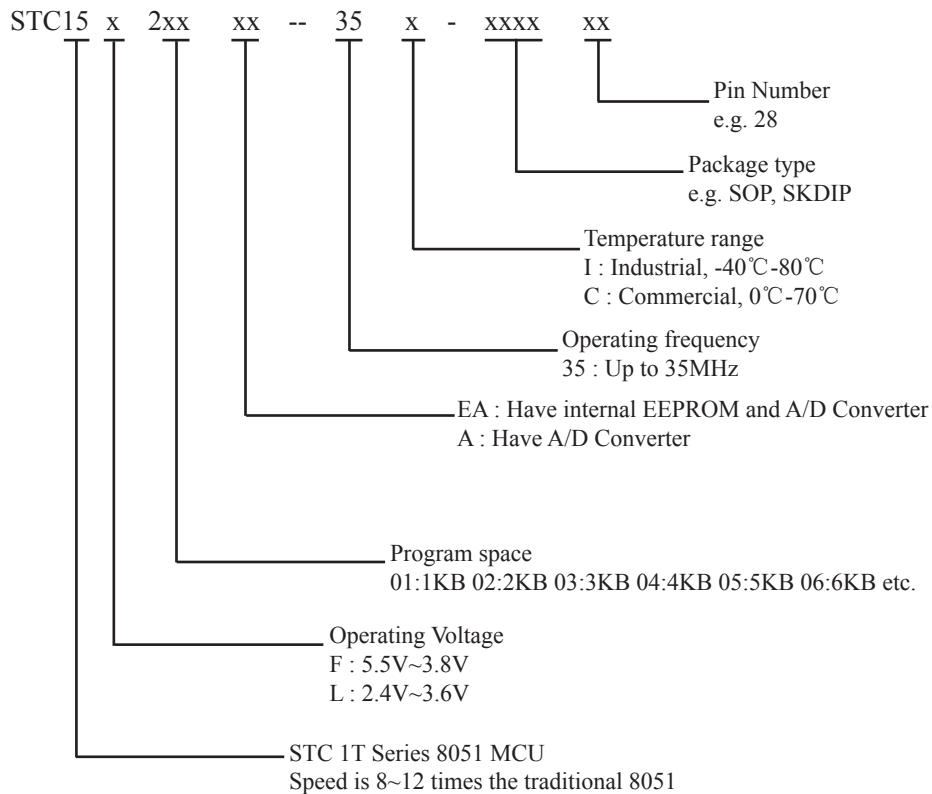
| COMMON DIMENSIONS         |       |       |       |
|---------------------------|-------|-------|-------|
| (UNITS OF MEASURE = INCH) |       |       |       |
| SYMBOL                    | MIN   | NOM   | MAX   |
| A                         | -     | -     | 0.175 |
| A1                        | 0.015 | -     | -     |
| A2                        | 0.125 | 0.13  | 0.135 |
| b                         | 0.016 | 0.018 | 0.020 |
| b1                        | 0.058 | 0.060 | 0.064 |
| C                         | 0.008 | 0.010 | 0.11  |
| D                         | 1.012 | 1.026 | 1.040 |
| E                         | 0.290 | 0.300 | 0.310 |
| E1                        | 0.245 | 0.250 | 0.255 |
| e                         | 0.090 | 0.100 | 0.110 |
| L                         | 0.120 | 0.130 | 0.140 |
| $\theta^0$                | 0     | -     | 15    |
| eA                        | 0.355 | 0.355 | 0.375 |
| S                         | -     | -     | 0.075 |

UNIT: INCH, 1 inch = 1000 mil

---

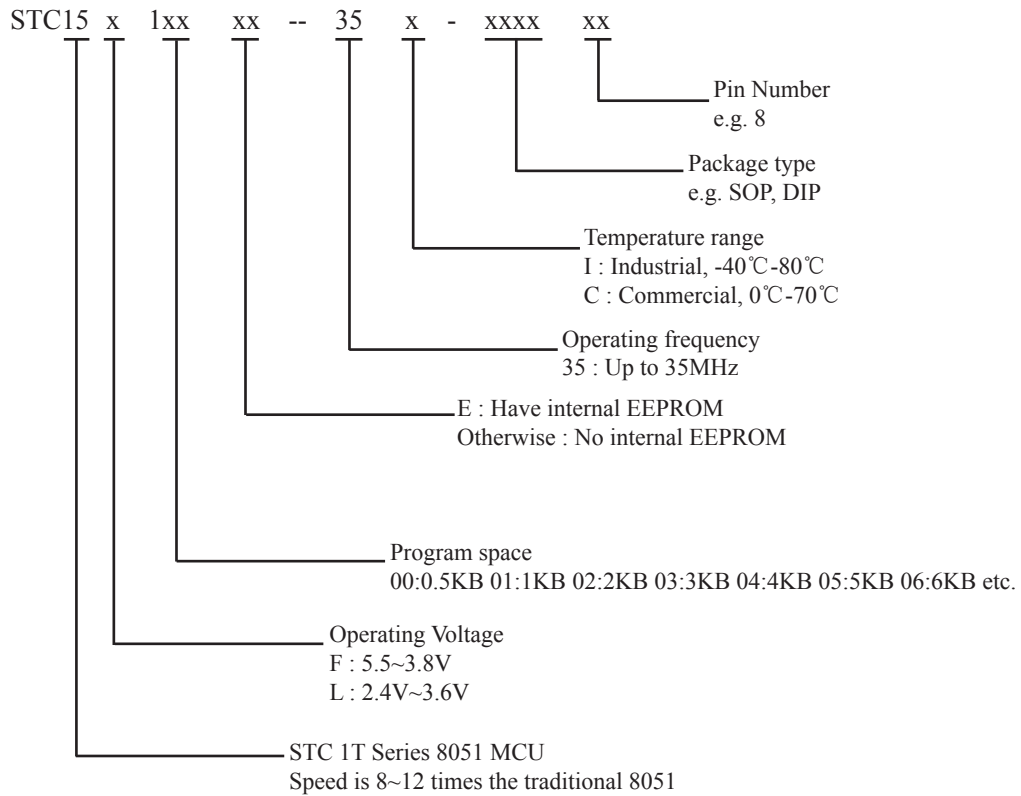
## 1.7 STC15Fxx series MCU naming rules

### 1.7.1 STC15F204EA series MCU naming rules



---

## 1.7.2 STC15F101E series MCU naming rules



## Chapter 2 Clock, Power Management, Reset

### 2.1 Clock

There is only one clock source—Internal RC oscillator available for STC15F204EA series.

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 | bit  | B7 | B6 | B5 | B4 | B3 | B2    | B1    | B0    |
| CLK_DIV  | 97H         | name | -  | -  | -  | -  | -  | 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

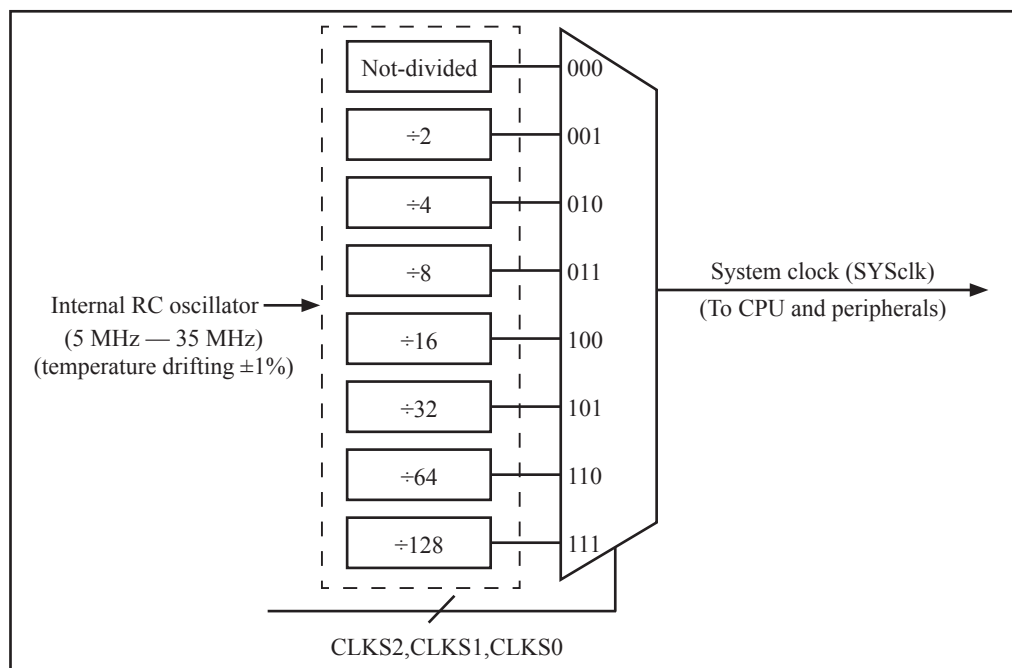
011 := The internal RC oscillator is set as the clock-in divided by 8

100 := The internal RC oscillator is set as the clock-in divided by 16

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)

| SFR name | Address | bit  | LSB |    |      |     |     |     |    |     |
|----------|---------|------|-----|----|------|-----|-----|-----|----|-----|
|          |         |      | B7  | B6 | B5   | B4  | B3  | B2  | B1 | B0  |
| PCON     | 87H     | name | -   | -  | LVDF | POF | GF1 | GF0 | 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).

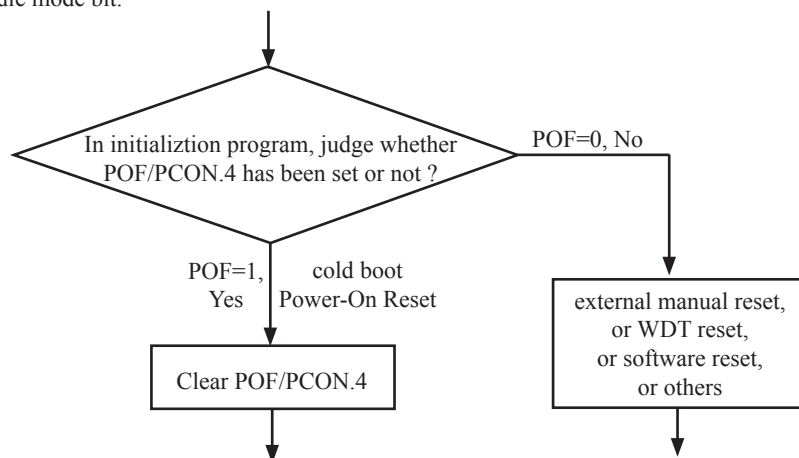
**POF** : Power-On flag. It is set by power-off-on action and can only be cleared by software.

**GF1** : General-purposed flag 1

**GF0** : General-purposed flag 0

**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 needs 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.

### CLK\_DIV register (Clock Divider)

| SFR Name | SFR Address | bit  | B7 | B6 | B5 | B4 | B3 | B2    | B1    | B0    |
|----------|-------------|------|----|----|----|----|----|-------|-------|-------|
| CLK_DIV  | 97H         | name | -  | -  | -  | -  | -  | 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

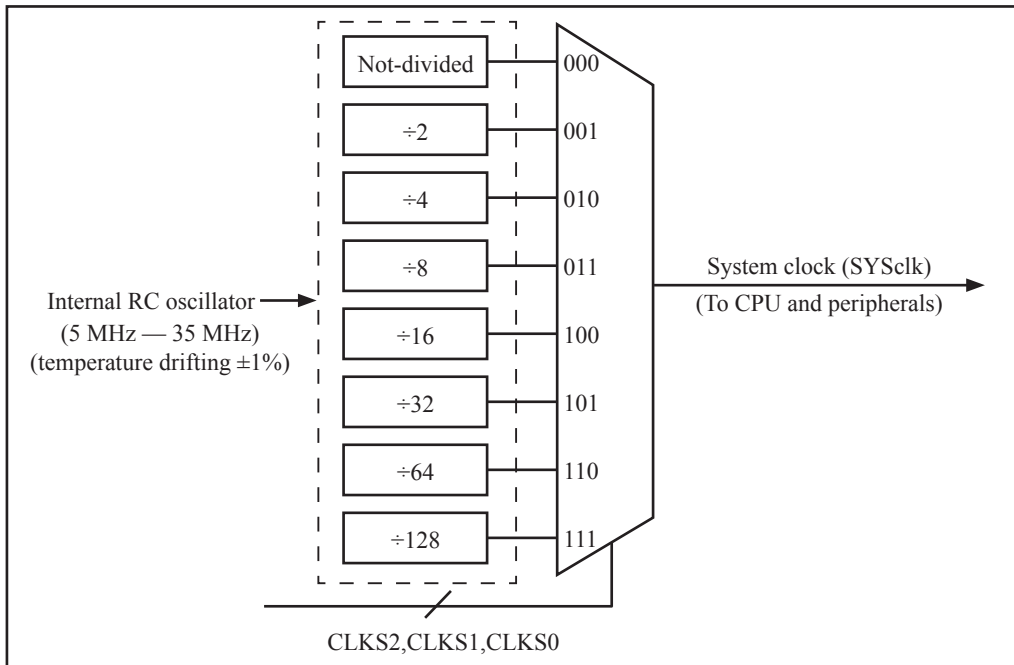
011 := The internal RC oscillator is set as the clock-in divided by 8

100 := The internal RC oscillator is set as the clock-in divided by 16

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



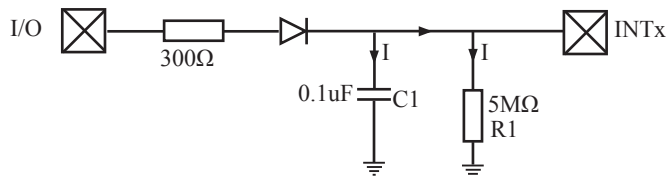
Clock Structure

---

### 2.2.3 Power Down (PD) Mode (Stop 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 interrupts INT0, INT1, INT2, INT3 and INT4. 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 external interrupts, the CPU will rework through jumping to related interrupt service routine. Before the CPU rework, the clock is blocked and counted until 64 in order for denouncing the unstable clock. To use external interrupts 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 external interrupts wake-up is used. When terminating Power-down by an interrupt, the wake up period is internally timed. At the negative edge (for INT0, INT1, INT2, INT3 and INT4) or positive edge (for INT0 and INT1) 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  $\mu$ s until after one of the following conditions has occurred: Start of code execution(after any type of reset), or Exit from power-down mode.

The following circuit can timing wake up MCU from power down mode when external interrupt sources do not exist



Operation step:

1. I/O ports are first configured to push-pull output(strong pull-up) mode
2. Writen 1s into ports I/O ports
3. the above circuit will charge the capacitor C1
4. Writen 0s into ports I/O ports, MCU will go into power-down mode
5. The above circuit will discharge. When the electricity of capacitor C1 has been discharged less than 0.8V, external interrupt INTx pin will generate a falling edge and wake up MCU from power-down mode automatically.

---

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

```
/*-----*/
/* --- STC MCU International Limited -----*/
/* --- STC 1T Series MCU wake up Power-Down mode Demo -----*/
/* 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 demonstrates that power-down mode or idle mode be woken-up by external interrupt, but is written in assembly language rather than C language.**

```

;*****
;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 STC15F204EA series, 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) Watch-Dog-Timer overflow
- (System-Invoked) MAX810-like Power-Up latency (~45mS)
- (System-Invoked) Low-Voltage detector acting

### 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 | bit  | B7    | B6   | B5    | B4       | B3 | B2  | B1  | B0  |
|-----------|-------------|------|-------|------|-------|----------|----|-----|-----|-----|
| IAP_CONTR | C7H         | name | 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

System will reset to AP address 0000H and begin running user application program code if

```
MOV    IAP_CONTR,    #00100000B
```

System will reset to ISP address 0000H and begin running system ISP monitor code if

```
MOV    IAP_CONTR,    #01100000B
```

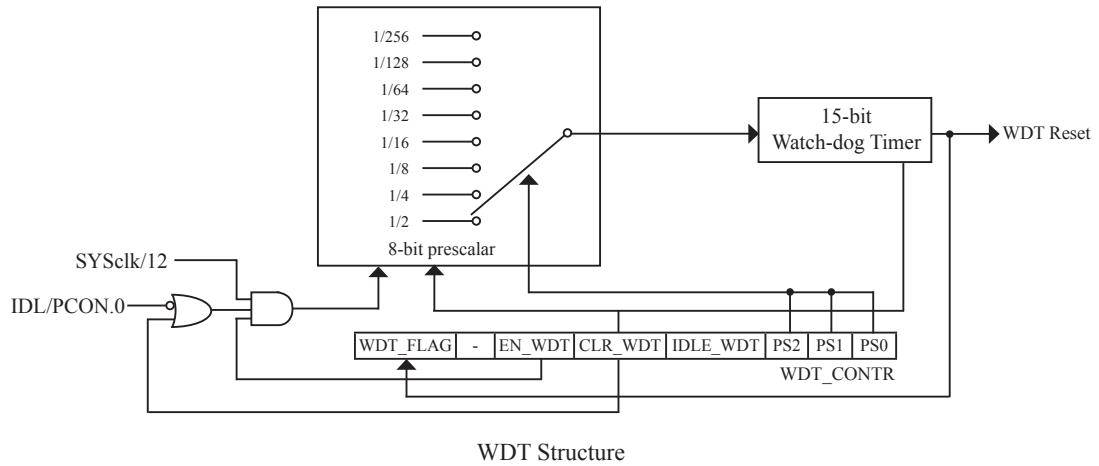
### 2.3.3 Power-On Reset (POR)

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 8192 clocks.

### 2.3.4 Watch-Dog-Timer

The watch dog timer in STC15F204EA series 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 STC15F204EA series out of control by any disturbance, that means the CPU can not run the software normally, then WDT may miss the "writing 1 to CLR\_WDT" and overflow will come. An overflow of Watch-Dog-Timer will generate a internal reset.



WDT\_CONTR: Watch-Dog-Timer Control Register

| SFR name  | Address | bit  | B7       | B6 | B5     | B4      | B3       | B2  | B1  | LSB<br>B0 |
|-----------|---------|------|----------|----|--------|---------|----------|-----|-----|-----------|
| WDT_CONTR | 0C1H    | name | 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:

$$\text{WDT overflow time} = (12 \times \text{Pre-scale} \times 32768) / \text{SYSclk}$$

The SYSclk is 20MHz in the table above.

If SYSclk is 12MHz, The WDT overflow time is :

$$\text{WDT overflow time} = (12 \times \text{Pre-scale} \times 32768) / 12000000 = \text{Pre-scale} \times 393216 / 12000000$$

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 demonstrates STC 1T Series MCU WDT.

```
;/*-----*/
;/* --- STC MCU International Limited -----*/
;/* --- STC 1T Series MCU WDT Demo -----*/
;/* 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 start-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\_Status

;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 STC15F204EA series. 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) voltgag for STC15F204EA series. When the VCC power drops down to the LVD voltage, the Low voltage Flag, LVDF 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 tables list all the low voltage detection threshold voltages under different degrees for STC15F204EA series .

5V device low voltage detection threshold voltages:

| -40 °C | 25 °C | 85 °C |
|--------|-------|-------|
| 4.74   | 4.64  | 4.60  |
| 4.41   | 4.32  | 4.27  |
| 4.14   | 4.05  | 4.00  |
| 3.90   | 3.82  | 3.77  |
| 3.69   | 3.61  | 3.56  |
| 3.51   | 3.43  | 3.38  |
| 3.36   | 3.28  | 3.23  |
| 3.21   | 3.14  | 3.09  |

User can select those voltages listed in above table as reset threshold voltages by STC ISP Writer/Programmer

3V device low voltage detection threshold voltages:

| -40 °C | 25 °C | 85 °C |
|--------|-------|-------|
| 3.11   | 3.08  | 3.09  |
| 2.85   | 2.82  | 2.83  |
| 2.63   | 2.61  | 2.61  |
| 2.44   | 2.42  | 2.43  |
| 2.29   | 2.26  | 2.26  |
| 2.14   | 2.12  | 2.12  |
| 2.01   | 2.00  | 2.00  |
| 1.90   | 1.89  | 1.89  |

User can select those voltages listed in above table as reset threshold voltages by STC ISP Writer/Programmer

---

Some SFRs related to Low voltage detection as shown below.

**PCON register** (Power Control Register)

|          |         |      |    |    |      |     |     |     |    | LSB |
|----------|---------|------|----|----|------|-----|-----|-----|----|-----|
| SFR name | Address | bit  | B7 | B6 | B5   | B4  | B3  | B2  | B1 | B0  |
| PCON     | 87H     | name | -  | -  | LVDF | POF | GF1 | GF0 | 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).

POF : Power-On flag. It is set by power-off-on action and can only cleared by software.

GF1 : General-purposed flag 1

GF0 : General-purposed flag 0

PD : Power-Down bit.

IDL : Idle mode bit.

**IE: Interrupt Enable Rsgister** (Address: 0A8H)

| (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** (Address: 0B8H)

| (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 STC15F204EA 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 STC15F204EA 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.

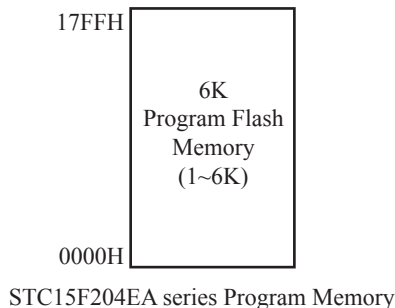
Data memory occupies a separate address space from program memory. In the STC15F204EA series, there are 256 bytes of internal scratch-pad RAM(SRAM).

### 3.1 Program Memory

Program memory is the memory which stores the program codes for the CPU to execute. There is 1~6Kbytes of flash memory embedded for program and data storage in STC15F204EA series. 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. IAP region is used to realize EEPROM function. In other words, the IAP capability of STC15F204EA series 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. STC15F204EA series MCU 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 STC15F204EA series 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.

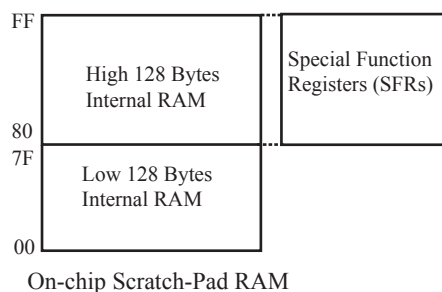
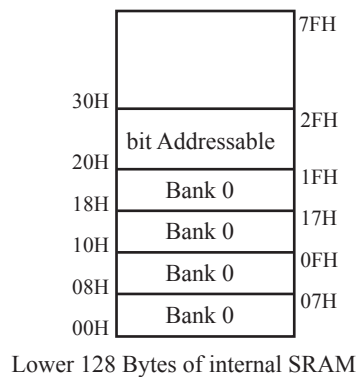


| Type            | Program Memory   |
|-----------------|------------------|
| STC15F/L201A/EA | 0000H~03FFH (1K) |
| STC15F/L202A/EA | 0000H~07FFH (2K) |
| STC15F/L203A/EA | 0000H~0BFFH (3K) |
| STC15F/L204A/EA | 0000H~0FFFH (4K) |
| STC15F/L205A/EA | 0000H~13FFH (5K) |
| IAP15F/L206A    | 0000H~17FFH (6K) |

## 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 STC15F204EA series. 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.



---

**PSW register**

LSB

| SFR name | Address | bit  | B7 | B6 | B5 | B4  | B3  | B2 | B1 | B0 |
|----------|---------|------|----|----|----|-----|-----|----|----|----|
| PSW      | D0H     | 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

STC15F204EA series 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 Port3, 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 ports on STC15F204EA series 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 ports default to quasi-bidirectional after reset. Each one has a Schmitt-triggered input for improved input noise rejection. The drive capability of each port is up to 20 mA. But recommend the whole chip's should be less than 90 mA.

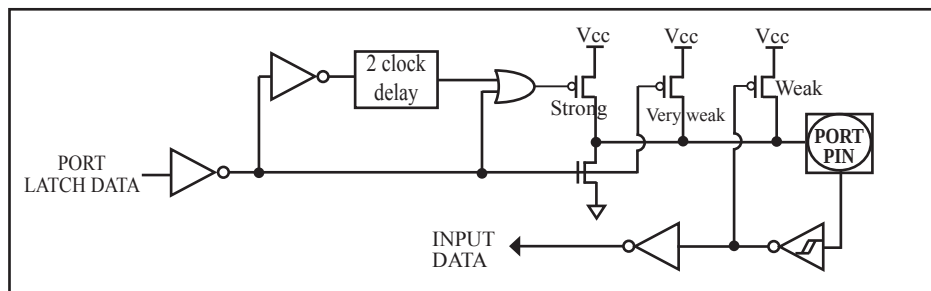
#### 4.1.1 Quasi-bidirectional I/O

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.

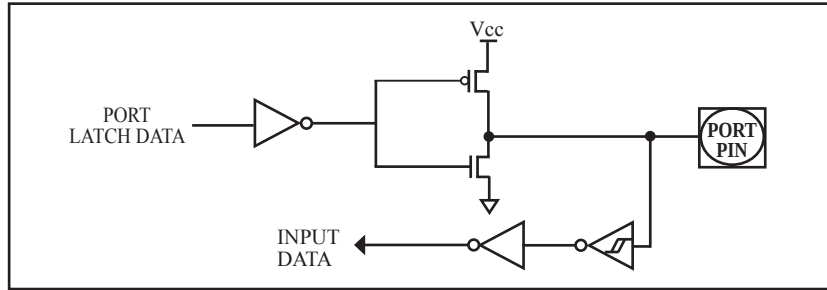


Quasi-bidirectional output

---

### 4.1.2 Push-pull Output

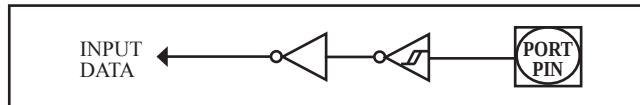
The push-pull output configuration has the same pull-down structure as both the open-drain and the quasi-bidirectional output modes, but provides a continuous strong pull-up when the port register contains a logic “1”. The push-pull mode may be used when more source current is needed from a port output.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

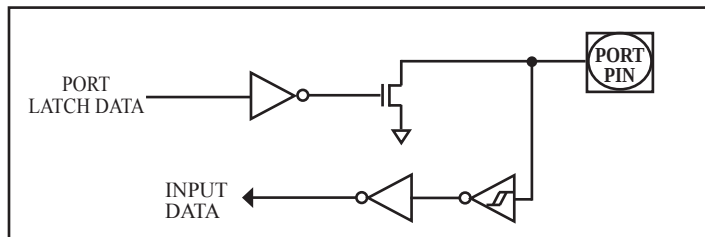
The input-only configuration is a Schmitt-triggered input without any pull-up resistors on the pin.



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 V<sub>CC</sub>. 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 STC15F204EA series 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 ~ 3 (port number), and n = 0 ~ 7 (port pin).

### P0 register

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

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

### P0M1 register

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

### P0M0 register

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

### P1 register

| SFR name | Address | bit  | B7   | B6   | B5   | B4   | B3   | B2   | B1   | B0   |
|----------|---------|------|------|------|------|------|------|------|------|------|
| P1       | 90H     | name | 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 could be set/cleared by CPU.

### P1M1 register

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

### P1M0 register

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

### P2 register

| SFR name | Address | bit  | B7   | B6   | B5   | B4   | B3   | B2   | B1   | B0   |
|----------|---------|------|------|------|------|------|------|------|------|------|
| P2       | A0H     | name | 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 could be set/cleared by CPU.

---

**P2M1 register**

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

**P2M0 register**

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

**P3 register**

| SFR name | Address | bit  | B7   | B6   | B5   | B4   | B3   | B2   | B1   | B0   |
|----------|---------|------|------|------|------|------|------|------|------|------|
| P3       | B0H     | name | 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 could be set/cleared by CPU.

**P3M1 register**

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

**P3M0 register**

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

---

### 4.3 I/O port application notes

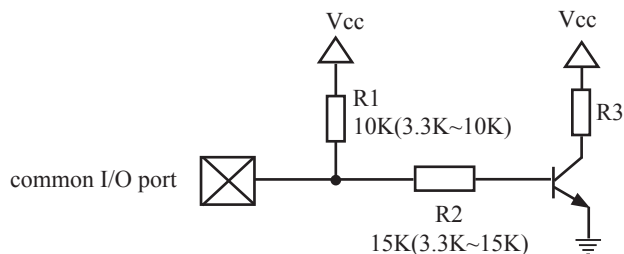
Traditional 8051 access I/O (signal transition or read status) timing is 12 clocks, STC15F204EA series MCU is 4 clocks. When you need to read an external signal, if internal output a rising edge signal, for the traditional 8051, this process is 12 clocks, you can read at once, but for STC15F204EA series MCU, this process is 4 clocks, when internal instructions is complete but external signal is not ready, so you must delay 1~2 nop operation.

Some I/O port connected to a PNP transistor, but no pul-up resistor. The correct access method is I/O port pull-up resistor and transistor base resistor should be consistent, or I/O port is set to a strongly push-pull output mode.

Using I/O port drive LED directly or matrix key scan, needs add a  $470\Omega$  to  $1K\Omega$  resistor to limit current.

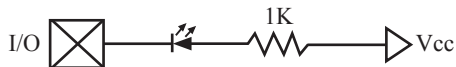
### 4.4 I/O port application

#### 4.4.1 Typical transistor control circuit



If I/O is configed as “weak” pull-up, you should add a external pull-up (3.3K~10K ohm). If no pull-up resistor R1, proposal to add a 15K ohm series resistor at least or config I/O as “push-pull” mode.

#### 4.4.2 Typical diode control circuit



For weak pull-up / quasi-bidirectional I/O, use sink current drive LED, current limiting resistor as greater than 1K ohm, minimum not less than 470 ohm.



For push-pull / strong pull-up I/O, use drive current drive LED.

---

### 4.4.3 3V/5V hybrid system

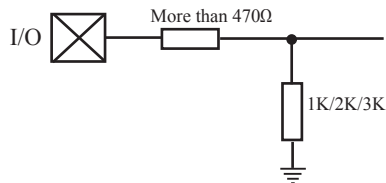
When STC15F204EA series 5V MCU connect to 3V peripherals. To prevent the device can not afford to 5V voltage, the corresponding I/O is set to open drain mode, disconnect the internal pull-up resistor, the corresponding I/O port add 10K ohm external pull-up resistor to the 3V device VCC, so high To 3V, low to 0V, which can proper functioning

When STC15F204EA series 3V MCU connect to 5V peripherals. To prevent the MCU can not afford to 5V voltage, if the corresponding I/O port as input port, the port may be in an isolation diode in series, isolated high-voltage part, the external signal is higher than MCU operating voltage, the diode cut-off, I/O I have been pulled high by the internal pull-up resistor; when the external signal is low, the diode conduction, I/O port voltage is limited to 0.7V, it's low signal to MCU.

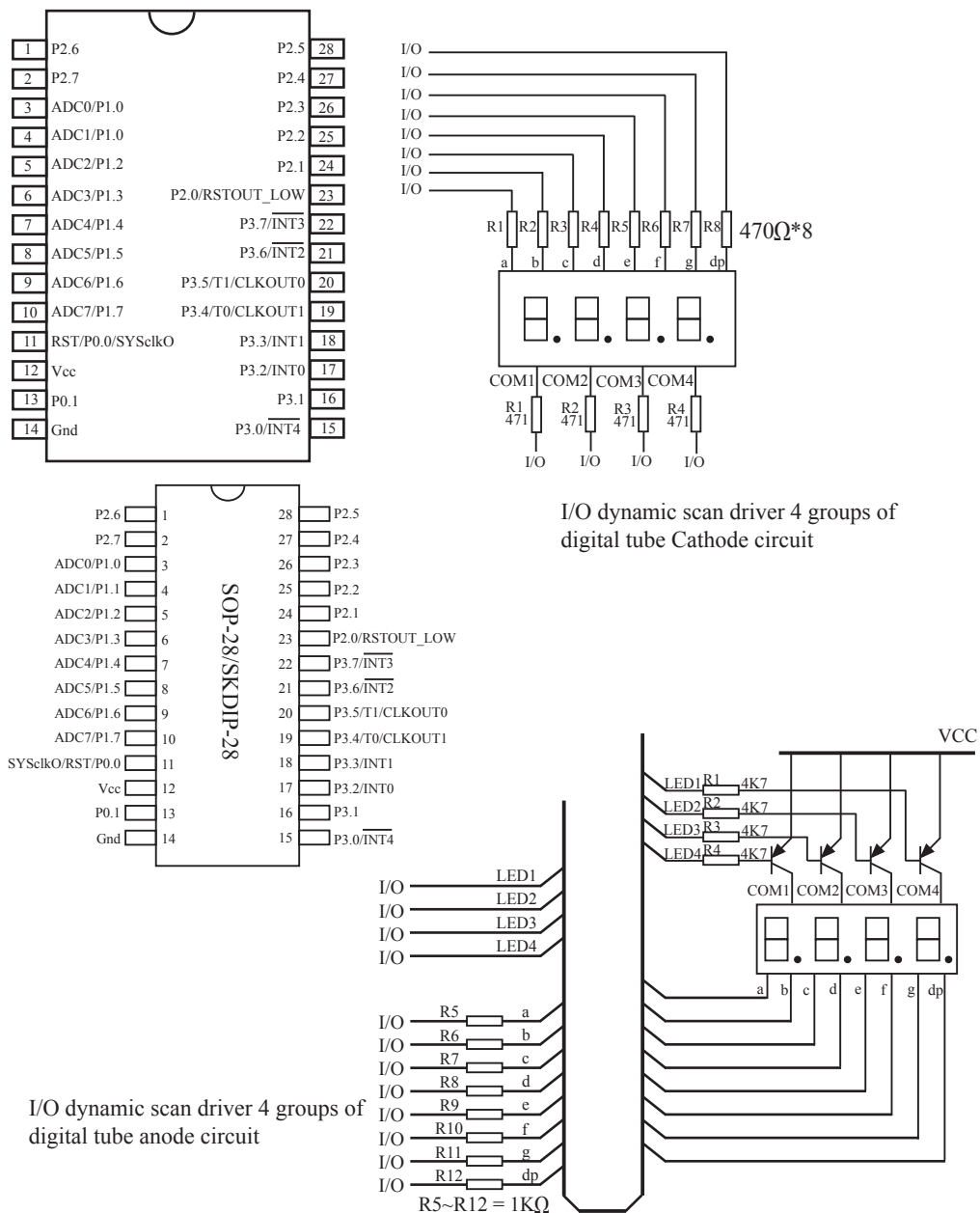


### 4.4.4 How to make I/O port low after MCU reset

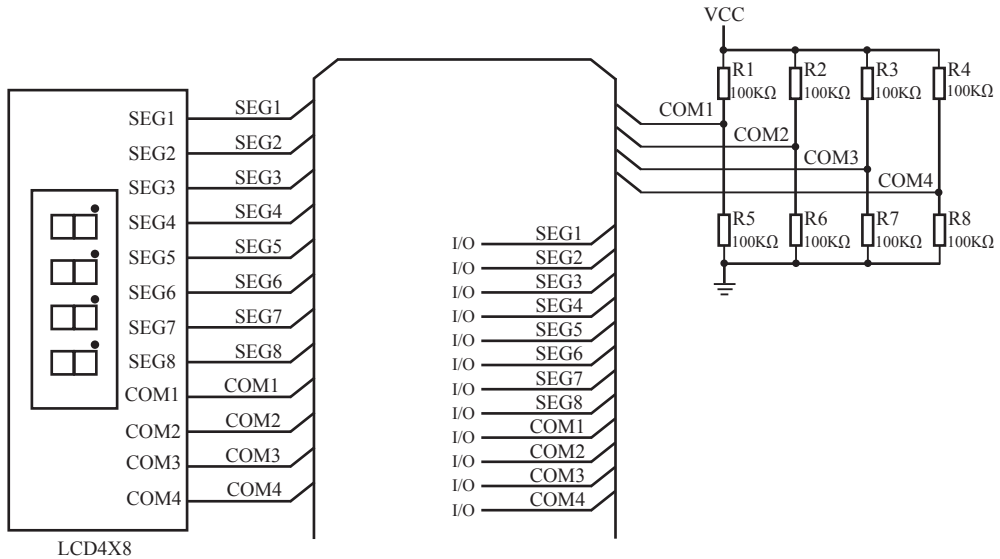
Traditional 8051 MCU power-on reset, the general I/O port are weak pull-high output, while many practical applications require I/O port remain low level after power-on reset, otherwise the system malfunction would be generated. For STC15F204EA series MCU, I/O port can add a pull-down resistor (1K/2K/3K), so that when power-on reset, although a weak internal pull-up to make MCU output high, but because of the limited capacity of the internal pull-up, it can not pull-high the pad, so this I/O port is low level after power-on reset. If the I/O port need to drive high, you can set the I/O model as the push-pull output mode, while the push-pull mode the drive current can be up to 20mA, so it can drive this I/O high.



#### 4.4.5 I/O drive LED application circuit



#### 4.4.6 I/O immediately drive LCD application circuit



How to light on the LCD pixels:

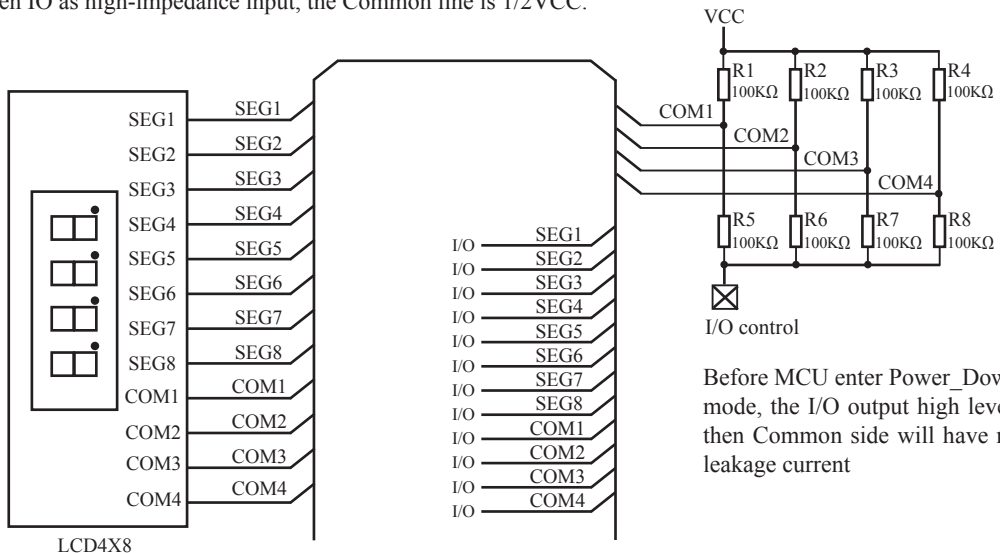
When the pixels corresponding COM-side and SEG-side voltage difference is greater than  $1/2V_{CC}$ , this pixel is lit, otherwise off

Control SEG-side (Segment) :

I/O direct drive Segment lines, control Segment output high-level (VCC) or low-level (0V).

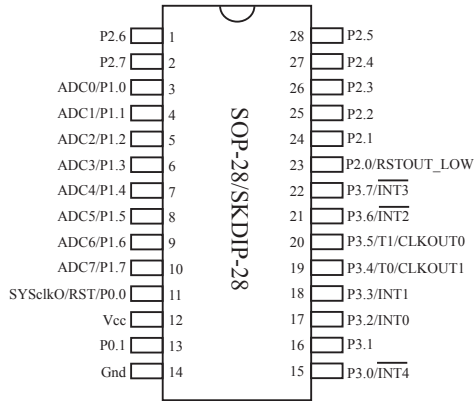
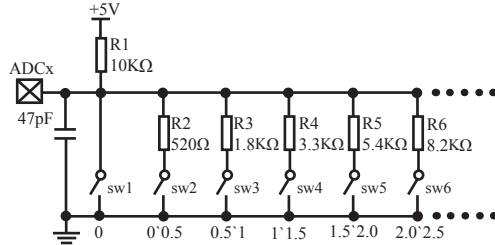
Control COM-side (Common) :

I/O port and two 100K dividing resistors jointly controlled Common line, when the IO output "0", the Common-line is low level (0V), when the IO push-pull output "1", the Common line is high level (VCC), when IO as high-impedance input, the Common line is  $1/2V_{CC}$ .



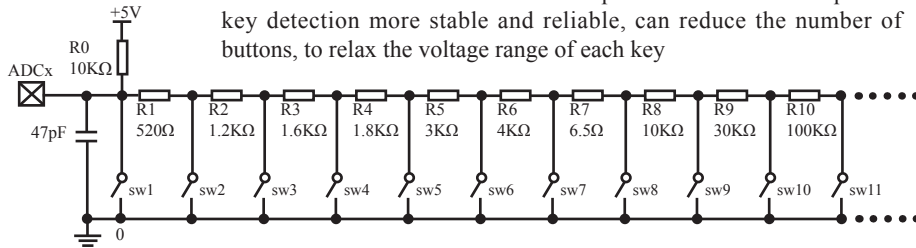
Before MCU enter Power\_Down mode, the I/O output high level, then Common side will have no leakage current

#### 4.4.7 Using A/D Conversion to scan key application circuit



This circuit can achieve a single key or combinational key scan, resistance needs to be configured according to actual needs.

This circuit uses 10 keys spaced partial pressure. For each key, the range of allowed error is  $\pm 0.25V$ , it can effectively avoid failure of key detection because of resistance or temperature drift. If the requested key detection is more stable and reliable, the number of buttons can be reduced to relax the voltage range of each key.



# 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<br>0000,0000    |                       |                       |                        |                        |                      |                       |                        | 0F7H |
| 0E8H |                   |                       |                       |                        |                        |                      |                       |                        | 0EFH |
| 0E0H | ACC<br>0000,0000  |                       |                       |                        |                        |                      |                       |                        | 0E7H |
| 0D8H |                   |                       |                       |                        |                        |                      |                       |                        | 0DFH |
| 0D0H | PSW<br>0000,00x0  |                       |                       |                        |                        |                      |                       |                        | 0D7H |
| 0C8H |                   |                       |                       |                        |                        |                      |                       |                        | 0CFH |
| 0C0H |                   | WDT_CONR<br>0x00,0000 | IAP_DATA<br>1111,1111 | IAP_ADDRH<br>0000,0000 | IAP_ADDRL<br>0000,0000 | IAP_CMD<br>xxxx,xx00 | IAP_TRIG<br>xxxx,xxxx | IAP_CONTR<br>0000,0000 | 0C7H |
| 0B8H | IP<br>x0x0,0000   |                       |                       | IRC_CLKO<br>0xxx,0xxx  | ADC_CONTR<br>0000,0000 | ADC_RES<br>0000,0000 | ADC_RESL<br>0000,0000 |                        | 0BFH |
| 0B0H | P3<br>1111,1111   | P3M1<br>0000,0000     | P3M0<br>0000,0000     |                        |                        |                      |                       |                        | 0B7H |
| 0A8H | IE<br>000x,0000   |                       |                       |                        |                        |                      |                       |                        | 0AFH |
| 0A0H | P2<br>1111,1111   |                       |                       |                        |                        |                      |                       | Don't use              | 0A7H |
| 098H |                   |                       |                       |                        |                        | P1ASF<br>0000,0000   | Don't use             | Don't use              | 09FH |
| 090H | P1<br>1111,1111   | P1M1<br>0000,0000     | P1M0<br>0000,0000     | P0M1<br>0000,0000      | P0M0<br>0000,0000      | P2M1<br>0000,0000    | P2M0<br>0000,0000     | CLK_DIV<br>xxxx,x000   | 097H |
| 088H | TCON<br>0000,0000 | TMOD<br>0000,0000     | TL0<br>0000,0000      | TL1<br>0000,0000       | TH0<br>0000,0000       | TH1<br>0000,0000     | AUXR<br>00xx,xxxx     | INT_CLKO<br>x000,xx00  | 08FH |
| 080H | P0<br>1111,1111   | SP<br>0000,0111       | DPL<br>0000,0000      | DPH<br>0000,0000       |                        |                      |                       | PCON<br>xx11,0000      | 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 |        |        |         |        |        |        |            | Value after Power-on or Reset |
|----------|-----------------------------------------------------|--------------------|---------|------------------------|--------|--------|---------|--------|--------|--------|------------|-------------------------------|
|          |                                                     |                    |         | MSB                    |        |        |         | LSB    |        |        |            |                               |
| 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                    |
|          | DPH                                                 | Data Pointer High  | 83H     |                        |        |        |         |        |        |        |            | 0000 0000B                    |
| PCON     |                                                     | Power Control      | 87H     | -                      | -      | LVDF   | POF     | GF1    | GF0    | PD     | IDL        | xx11 0000B                    |
| TCON     |                                                     | Timer Control      | 88H     | TF1                    | TR1    | TF0    | TR0     | IE1    | IT1    | IE0    | IT0        | 0000 0000B                    |
| TMOD     |                                                     | Timer Mode         | 89H     | GATE                   | C/T    | M1     | M0      | GATE   | C/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                |         |                        |        |        |         |        |        |        | 0000 0000B |                               |
|          |                                                     |                    | P17ASF  | P16ASF                 | P15ASF | P14ASF | P13ASF  | P12ASF | P11ASF | P10ASF |            |                               |
|          |                                                     |                    |         |                        |        |        |         |        |        |        |            |                               |
| 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 |        |        |          |           |      |      |      | Value after Power-on or Reset |
|-----------|----------------------------------|---------|------------------------|--------|--------|----------|-----------|------|------|------|-------------------------------|
|           |                                  |         | MSB                    |        |        |          | LSB       |      |      |      |                               |
| 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 incremented before data is stored during PUSH and CALL executions. While the stack may reside anywhere 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

| SFR name | Address | bit  | B7 | B6 | B5 | B4  | B3  | B2 | B1 | B0 |
|----------|---------|------|----|----|----|-----|-----|----|----|----|
| PSW      | D0H     | 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 | bit  | B7    | B6    | B5 | B4 | B3 | B2 | B1 | B0 |
|----------|-------------|------|-------|-------|----|----|----|----|----|----|
| AUXR     | 8EH         | name | 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 | bit  | B7 | B6  | B5  | B4  | B3 | B2 | B1     | B0     |
|----------|-------------|------|----|-----|-----|-----|----|----|--------|--------|
| INT_CLKO | 8FH         | name | -  | EX4 | EX3 | EX2 | -  | -  | T1CLKO | T0CLKO |

EX4

- 0 := Disable  $\overline{\text{INT4}}$  interrupt function.
- 1 := Enable  $\overline{\text{INT4}}$  interrupt function.

EX3

- 0 := Disable  $\overline{\text{INT3}}$  interrupt function.
- 1 := Enable  $\overline{\text{INT3}}$  interrupt function.

EX2

- 0 := Disable  $\overline{\text{INT2}}$  interrupt function.
- 1 := Enable  $\overline{\text{INT2}}$  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 specifying 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 STC15F204EA series |
|------------------------------|----------------------------------------------|------|---------------------------------------|----------------------------------------|
| <b>ARITHMETIC OPERATIONS</b> |                                              |      |                                       |                                        |
| ADD A, Rn                    | Add register to Accumulator                  | 1    | 12                                    | 2                                      |
| ADD A, direct                | Add direct 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 with 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                | Subtract 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                       | Divide 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 STC15F204EA series |
|---------------------------|--------------------------------------------|------|---------------------------------------|----------------------------------------|
| <b>LOGICAL OPERATIONS</b> |                                            |      |                                       |                                        |
| ANL A, Rn                 | AND Register to Accumulator                | 1    | 12                                    | 2                                      |
| ANL A, direct             | AND direct byte 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 STC15F204EA series |
|----------------------|------------------------------------------------|------|---------------------------------------|----------------------------------------|
| <b>DATA TRANSFER</b> |                                                |      |                                       |                                        |
| 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 immediate 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 STC15F204EA series |
|--------------------------------------|-----------------------------------------------------|------|---------------------------------------|----------------------------------------|
| <b>BOOLEAN VARIABLE MANIPULATION</b> |                                                     |      |                                       |                                        |
| 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                                      |
| <b>PROGRAM BRANCHING</b>             |                                                     |      |                                       |                                        |
| 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 for Standard 8051 MCU

### 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 successively 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 executing the 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  
 $(PC) \leftarrow (PC) + 2$   
 $(SP) \leftarrow (SP) + 1$   
 $((SP)) \leftarrow (PC_{7-0})$   
 $(SP) \leftarrow (SP) + 1$   
 $((SP)) \leftarrow (PC_{15-8})$   
 $(PC_{10-0}) \leftarrow \text{page address}$

### 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 occurred.

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  
 $(A) \leftarrow (A) + (Rn)$ **ADD A,direct****Bytes:** 2**Cycles:** 1**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 1 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 1 |
|---|---|---|---|

|                |
|----------------|
| direct address |
|----------------|

**Operation:** ADD  
 $(A) \leftarrow (A) + (\text{direct})$ **ADD A,@Ri****Bytes:** 1**Cycles:** 1**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 1 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | i |
|---|---|---|---|

**Operation:** ADD  
 $(A) \leftarrow (A) + ((Ri))$ **ADD A,#data****Bytes:** 2**Cycles:** 1**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 1 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 0 |
|---|---|---|---|

|                |
|----------------|
| immediate data |
|----------------|

**Operation:** ADD  
 $(A) \leftarrow (A) + \#data$ **ADDC A,<src-byte>**

---

**Function:** Add with Carry**Description:** ADC 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 occurred.

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, ADC 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  
 $(A) \leftarrow (A) + (C) + (Rn)$ **ADDC A,direct****Bytes:** 2**Cycles:** 1**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 1 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 1 |
|---|---|---|---|

|                |
|----------------|
| direct address |
|----------------|

**Operation:** ADDC  
 $(A) \leftarrow (A) + (C) + (\text{direct})$ **ADDC A,@Ri****Bytes:** 1**Cycles:** 1**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 1 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | i |
|---|---|---|---|

**Operation:** ADDC  
 $(A) \leftarrow (A) + (C) + ((Ri))$ **ADDC A,#data****Bytes:** 2**Cycles:** 1**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 1 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 0 |
|---|---|---|---|

|                |
|----------------|
| immediate data |
|----------------|

**Operation:** ADDC  
 $(A) \leftarrow (A) + (C) + \#data$ 

---

**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  
 $(PC) \leftarrow (PC) + 2$   
 $(PC_{10-0}) \leftarrow \text{page address}$

---

## 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 PI, #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  
 $(A) \leftarrow (A) \wedge (Rn)$

### ANL A,direct

**Bytes:** 2

**Cycles:** 1

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 1 |
|---|---|---|---|

|                |
|----------------|
| direct address |
|----------------|

**Operation:** ANL  
 $(A) \leftarrow (A) \wedge (\text{direct})$

### ANL A,@Ri

**Bytes:** 1

**Cycles:** 1

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | i |
|---|---|---|---|

**Operation:** ANL  
 $(A) \leftarrow (A) \wedge ((Ri))$

---

**ANL A,#data****Bytes:** 2**Cycles:** 1**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 0 |
|---|---|---|---|

|                |  |  |  |
|----------------|--|--|--|
| immediate data |  |  |  |
|----------------|--|--|--|

**Operation:** ANL  
 $(A) \leftarrow (A) \wedge \#data$ **ANL direct,A****Bytes:** 2**Cycles:** 1**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 1 | 0 |
|---|---|---|---|

|                |  |  |  |
|----------------|--|--|--|
| direct address |  |  |  |
|----------------|--|--|--|

**Operation:** ANL  
 $(direct) \leftarrow (direct) \wedge (A)$ **ANL direct,#data****Bytes:** 3**Cycles:** 2**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 1 | 1 |
|---|---|---|---|

|                |  |  |  |
|----------------|--|--|--|
| direct address |  |  |  |
|----------------|--|--|--|

|                |  |  |  |
|----------------|--|--|--|
| immediate data |  |  |  |
|----------------|--|--|--|

**Operation:** ANL  
 $(direct) \leftarrow (direct) \wedge \#data$ **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 affected*. No other flgs 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  
 $(C) \leftarrow (C) \wedge (bit)$

---

**ANL C, /bit****Bytes:** 2**Cycles:** 2**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 0 | 1 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
|---|---|---|---|

|             |
|-------------|
| bit address |
|-------------|

**Operation:** ADD $(C) \leftarrow (C) \wedge \overline{(\text{bit})}$ 

---

**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 P1, 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:**  $(PC) \leftarrow (PC) + 3$ 

IF (A) &lt;&gt; (direct)

THEN

 $(PC) \leftarrow (PC) + \text{relative offset}$ 

IF (A) &lt; (direct)

THEN

 $(C) \leftarrow 1$ 

ELSE

 $(C) \leftarrow 0$

---

**CJNE A,#data,rel****Bytes:** 3**Cycles:** 2**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 0 | 1 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 1 |
|---|---|---|---|







**Operation:**  $(PC) \leftarrow (PC) + 3$   
IF  $(A) \neq (data)$   
THEN  
     $(PC) \leftarrow (PC) + \text{relative offset}$   
IF  $(A) < (data)$   
THEN  
     $(C) \leftarrow 1$   
ELSE  
     $(C) \leftarrow 0$

**CJNE Rn,#data,rel****Bytes:** 3**Cycles:** 2**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 0 | 1 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 1 | r | r | r |
|---|---|---|---|







**Operation:**  $(PC) \leftarrow (PC) + 3$   
IF  $(Rn) \neq (data)$   
THEN  
     $(PC) \leftarrow (PC) + \text{relative offset}$   
IF  $(Rn) < (data)$   
THEN  
     $(C) \leftarrow 1$   
ELSE  
     $(C) \leftarrow 0$

**CJNE @Ri,#data,rel****Bytes:** 3**Cycles:** 2**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 0 | 1 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | i |
|---|---|---|---|







**Operation:**  $(PC) \leftarrow (PC) + 3$   
IF  $((Ri)) \neq (data)$   
THEN  
     $(PC) \leftarrow (PC) + \text{relative offset}$   
IF  $((Ri)) < (data)$   
THEN  
     $(C) \leftarrow 1$   
ELSE  
     $(C) \leftarrow 0$

---

## CLR A

---

**Function:** Clear Accumulator

**Description:** The Accumulator 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  $\overline{\quad}$   
(A) ← (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  $\overline{\quad}$   
(C) ← (C)

---

## CPL bit

---

**Bytes:** 2

**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
|---|---|---|---|---|---|---|---|

|             |
|-------------|
| bit address |
|-------------|

**Operation:** CPL  $\overline{\quad}$   
(bit) ← (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  $[(A_{3-0}) > 9] \vee [(AC) = 1]$   
   THEN  $(A_{3-0}) \leftarrow (A_{3-0}) + 6$   
       AND  
 IF  $[(A_{7-4}) > 9] \vee [(C) = 1]$   
   THEN  $(A_{7-4}) \leftarrow (A_{7-4}) + 6$

## 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  
 $(A) \leftarrow (A) - 1$

## DEC Rn

**Bytes:** 1  
**Cycles:** 1  
**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 1 | r | r | r |

  
**Operation:** DEC  
 $(Rn) \leftarrow (Rn) - 1$

---

---

**DEC direct****Bytes:** 2**Cycles:** 1**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 1 |
|---|---|---|---|

|                |
|----------------|
| direct address |
|----------------|

**Operation:** DEC  
 $(\text{direct}) \leftarrow (\text{direct}) - 1$ **DEC @Ri****Bytes:** 1**Cycles:** 1**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | i |
|---|---|---|---|

**Operation:** DEC  
 $((Ri)) \leftarrow ((Ri)) - 1$ 

---

**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  $251 = (13 \times 18) + 17$ . Carry and OV will both be cleared.**Bytes:** 1**Cycles:** 4**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 0 |
|---|---|---|---|

**Operation:** DIV  
$$\begin{matrix} (A)_{15-8} \\ (B)_{7-0} \end{matrix} \leftarrow (A)/(B)$$

---

**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 affected. 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
TOGGLE: CPL P1.7
        DJNZ R2, TOGGLE
```

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  
 $(PC) \leftarrow (PC) + 2$   
 $(Rn) \leftarrow (Rn) - 1$   
IF  $(Rn) > 0$  or  $(Rn) < 0$   
THEN  
 $(PC) \leftarrow (PC) + rel$

**DJNZ direct, rel**

**Bytes:** 3

**Cycles:** 2

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 1 | 0 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 1 |
|---|---|---|---|

|                |
|----------------|
| direct address |
|----------------|

|              |
|--------------|
| rel. address |
|--------------|

---

**Operation:** DJNZ  
 $(PC) \leftarrow (PC) + 2$   
 $(direct) \leftarrow (direct) - 1$   
IF  $(direct) > 0$  or  $(direct) < 0$   
THEN  
 $(PC) \leftarrow (PC) + rel$

## 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 (01111110B). 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  
 $(A) \leftarrow (A) + 1$

## INC Rn

**Bytes:** 1

**Cycles:** 1

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 1 | r | r | r |
|---|---|---|---|

**Operation:** INC  
 $(Rn) \leftarrow (Rn) + 1$

## INC direct

**Bytes:** 2

**Cycles:** 1

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 1 |
|---|---|---|---|

|                |
|----------------|
| direct address |
|----------------|

**Operation:** INC  
 $(direct) \leftarrow (direct) + 1$

---

**INC @Ri****Bytes:** 1**Cycles:** 1**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | i |
|---|---|---|---|

**Operation:** INC  
 $((Ri)) \leftarrow ((Ri)) + 1$ 

---

**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  
 $(DPTR) \leftarrow (DPTR) + 1$ 

---

**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  
 $(PC) \leftarrow (PC) + 3$   
IF (bit) = 1  
THEN  
 $(PC) \leftarrow (PC) + rel$

---

## 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 will 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  
 $(PC) \leftarrow (PC) + 3$   
IF (bit) = 1  
THEN  
    (bit)  $\leftarrow$  0  
     $(PC) \leftarrow (PC) + \text{rel}$

## 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  
 $(PC) \leftarrow (PC) + 2$   
IF (C) = 1  
THEN  
     $(PC) \leftarrow (PC) + \text{rel}$

---

## 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)  $\leftarrow$  (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)  $\leftarrow$  (PC) + 3  
IF (bit) = 0  
THEN (PC)  $\leftarrow$  (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  
 $(PC) \leftarrow (PC) + 2$   
IF  $(C) = 0$   
THEN  $(PC) \leftarrow (PC) + rel$

---

**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  
 $(PC) \leftarrow (PC) + 2$   
IF  $(A) \neq 0$   
THEN  $(PC) \leftarrow (PC) + rel$

---

**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  
 $(PC) \leftarrow (PC) + 2$   
IF  $(A) = 0$   
THEN  $(PC) \leftarrow (PC) + \text{rel}$

---

**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  
 $(PC) \leftarrow (PC) + 3$   
 $(SP) \leftarrow (SP) + 1$   
 $((SP)) \leftarrow (PC_{7-0})$   
 $(SP) \leftarrow (SP) + 1$   
 $((SP)) \leftarrow (PC_{15-8})$   
 $(PC) \leftarrow \text{addr}_{15-0}$

---

## 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)  $\leftarrow$  addr<sub>15:0</sub>

---

## 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, @R1      ;B <= 10H
MOV    @R1, P1     ;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)  $\leftarrow$  (Rn)

---

**\*MOV A,direct****Bytes:** 2**Cycles:** 1**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 1 | 1 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 1 |
|---|---|---|---|

|                |
|----------------|
| direct address |
|----------------|

**Operation:** MOV  
(A) $\leftarrow$  (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)  $\leftarrow$  ((Ri))**MOV A,#data****Bytes:** 2**Cycles:** 1**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 0 |
|---|---|---|---|

|                |
|----------------|
| immediate data |
|----------------|

**Operation:** MOV  
(A) $\leftarrow$  #data**MOV Rn,A****Bytes:** 1**Cycles:** 1**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 1 | r | r | r |
|---|---|---|---|

**Operation:** MOV  
(Rn) $\leftarrow$  (A)**MOV Rn,direct****Bytes:** 2**Cycles:** 2**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 0 | 1 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 1 | r | r | r |
|---|---|---|---|

|              |
|--------------|
| direct addr. |
|--------------|

**Operation:** MOV  
(Rn) $\leftarrow$  (direct)**MOV Rn,#data****Bytes:** 2**Cycles:** 1**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 1 | r | r | r |
|---|---|---|---|

|                |
|----------------|
| immediate data |
|----------------|

**Operation:** MOV  
(Rn)  $\leftarrow$  #data

---

**MOV direct, A****Bytes:** 2**Cycles:** 1**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 1 |
|---|---|---|---|

|                |
|----------------|
| direct address |
|----------------|

**Operation:** MOV  
(direct)  $\leftarrow$  (A)**MOV direct, Rn****Bytes:** 2**Cycles:** 2**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 1 | r | r | r |
|---|---|---|---|

|                |
|----------------|
| direct address |
|----------------|

**Operation:** MOV  
(direct)  $\leftarrow$  (Rn)**MOV direct, direct****Bytes:** 3**Cycles:** 2**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 1 |
|---|---|---|---|

|                 |
|-----------------|
| dir.addr. (src) |
|-----------------|

**Operation:** MOV  
(direct)  $\leftarrow$  (direct)**MOV direct, @Ri****Bytes:** 2**Cycles:** 2**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | i |
|---|---|---|---|

|              |
|--------------|
| direct addr. |
|--------------|

**Operation:** MOV  
(direct)  $\leftarrow$  ((Ri))**MOV direct, #data****Bytes:** 3**Cycles:** 2**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 1 |
|---|---|---|---|

|                |
|----------------|
| direct address |
|----------------|

**Operation:** MOV  
(direct)  $\leftarrow$  #data**MOV @Ri, A****Bytes:** 1**Cycles:** 1**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | i |
|---|---|---|---|

**Operation:** MOV  
((Ri))  $\leftarrow$  (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)  $\leftarrow$  #data<sub>15-0</sub>  
DPH DPL  $\leftarrow$  #data<sub>15-8</sub> #data<sub>7-0</sub>

---

## 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 defined 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)  $\leftarrow$  ((A)+(DPTR))

---

**MOVC A,@A+PC****Bytes:** 1**Cycles:** 2**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 1 | 1 |
|---|---|---|---|

**Operation:** MOVC  
(PC)  $\leftarrow$  (PC)+1  
(A)  $\leftarrow$  ((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)  $\leftarrow$  ((Ri))

---

**MOVX A,@DPTR****Bytes:** 1**Cycles:** 2**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 1 | 1 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
|---|---|---|---|

**Operation:** MOVX  
 $(A) \leftarrow ((DPTR))$ **MOVX @Ri, A****Bytes:** 1**Cycles:** 2**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 1 | i |
|---|---|---|---|

**Operation:** MOVX  
 $((Ri)) \leftarrow (A)$ **MOVX @DPTR, A****Bytes:** 1**Cycles:** 2**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
|---|---|---|---|

**Operation:** MOVX  
 $(DPTR) \leftarrow (A)$ 

---

**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  
 $(A)_{7-0} \leftarrow (A) \times (B)$   
 $(B)_{15-8}$

---

## 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 1 of output Port 1.

---

**ORL A,Rn****Bytes:** 1**Cycles:** 1**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 1 | r | r | r |
|---|---|---|---|

**Operation:** ORL  
 $(A) \leftarrow (A) \vee (Rn)$ **ORL A,direct****Bytes:** 2**Cycles:** 1**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 1 |
|---|---|---|---|

|                |
|----------------|
| direct address |
|----------------|

**Operation:** ORL  
 $(A) \leftarrow (A) \vee (\text{direct})$ **ORL A,@Ri****Bytes:** 1**Cycles:** 1**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | i |
|---|---|---|---|

**Operation:** ORL  
 $(A) \leftarrow (A) \vee ((Ri))$ **ORL A,#data****Bytes:** 2**Cycles:** 1**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 0 |
|---|---|---|---|

|                |
|----------------|
| immediate data |
|----------------|

**Operation:** ORL  
 $(A) \leftarrow (A) \vee \#data$ **ORL direct, A****Bytes:** 2**Cycles:** 1**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 1 | 0 |
|---|---|---|---|

|                |
|----------------|
| direct address |
|----------------|

**Operation:** ORL  
 $(\text{direct}) \leftarrow (\text{direct}) \vee (A)$ **ORL direct, #data****Bytes:** 3**Cycles:** 2**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 1 | 1 |
|---|---|---|---|

|                |
|----------------|
| direct address |
|----------------|

|                |
|----------------|
| immediate data |
|----------------|

**Operation:** ORL  
 $(\text{direct}) \leftarrow (\text{direct}) \vee \#data$

---

**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  
 $(C) \leftarrow (C) \vee (\text{bit})$

**ORL C, /bit**

**Bytes:** 2

**Cycles:** 2

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 0 | 1 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
|---|---|---|---|

|             |
|-------------|
| bit address |
|-------------|

**Operation:** ORL  
 $(C) \leftarrow (C) \vee \overline{(\text{bit})}$

---

**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  
 $(\text{direct}) \leftarrow ((SP))$   
 $(SP) \leftarrow (SP) - 1$

---

**PUSH direct**

---

**Function:** Push onto stack

**Description:** The Stack Pointer is incremented by one. The contents of the indicated variable is 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  
 $(SP) \leftarrow (SP) + 1$   
 $((SP)) \leftarrow (\text{direct})$

---

**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  
 $(PC_{15-8}) \leftarrow ((SP))$   
 $(SP) \leftarrow (SP) - 1$   
 $(PC_{7-0}) \leftarrow ((SP))$   
 $(SP) \leftarrow (SP) - 1$

---

## 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  
 $(A_{n+1}) \leftarrow (A_n) \quad n = 0-6$   
 $(A_0) \leftarrow (A_7)$

---

## 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  
 $(A_{n+1}) \leftarrow (A_n) \quad n = 0-6$   
 $(A_0) \leftarrow (C)$   
 $(C) \leftarrow (A_7)$

---

## 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  
 $(A_n) \leftarrow (A_{n+1}) \quad n = 0 - 6$   
 $(A_7) \leftarrow (A_0)$

---

## 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  
 $(A_{n+1}) \leftarrow (A_n) \quad n = 0-6$   
 $(A_7) \leftarrow (C)$   
 $(C) \leftarrow (A_0)$

---

**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  
 $(A) \leftarrow (A) - (C) - (Rn)$

**SUBB A, direct**

**Bytes:** 2

**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|---|

|                |
|----------------|
| direct address |
|----------------|

**Operation:** SUBB  
 $(A) \leftarrow (A) - (C) - (\text{direct})$

**SUBB A, @Ri**

**Bytes:** 1

**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 0 | 1 | 1 | i |
|---|---|---|---|---|---|---|---|

**Operation:** SUBB  
 $(A) \leftarrow (A) - (C) - ((Ri))$

---

**SUBB A, #data****Bytes:** 2**Cycles:** 1**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
|---|---|---|---|---|---|---|---|

|                |
|----------------|
| immediate data |
|----------------|

**Operation:** SUBB  
 $(A) \leftarrow (A) - (C) - \#data$ 

---

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

---

**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  
 $(A) \longleftrightarrow (Rn)$ 

---

**XCH A, direct****Bytes:** 2**Cycles:** 1**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|---|

|                |
|----------------|
| direct address |
|----------------|

**Operation:** XCH  
 $(A) \longleftrightarrow (\text{direct})$

---

**XCH A, @Ri****Bytes:** 1**Cycles:** 1**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 1 | 0 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | i |
|---|---|---|---|

**Operation:** XCH  
(A)  $\longleftrightarrow$  ((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<sub>3-0</sub>)  $\longleftrightarrow$  (Ri<sub>3-0</sub>)**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 value 69H (01101001B).

When the destination is a directly addressed byte, this instruction can complement combination 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 output Port 1.

---

**XRL A, Rn****Bytes:** 1**Cycles:** 1**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 1 | r | r | r |
|---|---|---|---|

**Operation:** XRL  
 $(A) \leftarrow (A) \hat{\wedge} (Rn)$ **XRL A, direct****Bytes:** 2**Cycles:** 1**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 1 |
|---|---|---|---|

|                |
|----------------|
| direct address |
|----------------|

**Operation:** XRL  
 $(A) \leftarrow (A) \hat{\wedge} (\text{direct})$ **XRL A, @Ri****Bytes:** 1**Cycles:** 1**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | i |
|---|---|---|---|

**Operation:** XRL  
 $(A) \leftarrow (A) \hat{\wedge} ((Ri))$ **XRL A, #data****Bytes:** 2**Cycles:** 1**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 0 |
|---|---|---|---|

|                |
|----------------|
| immediate data |
|----------------|

**Operation:** XRL  
 $(A) \leftarrow (A) \hat{\wedge} \#data$ **XRL direct, A****Bytes:** 2**Cycles:** 1**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 1 | 0 |
|---|---|---|---|

|                |
|----------------|
| direct address |
|----------------|

**Operation:** XRL  
 $(\text{direct}) \leftarrow (\text{direct}) \hat{\wedge} (A)$ **XRL direct, #dataw****Bytes:** 3**Cycles:** 2**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 1 | 1 |
|---|---|---|---|

|                |
|----------------|
| direct address |
|----------------|

|                |
|----------------|
| immediate data |
|----------------|

**Operation:** XRL  
 $(\text{direct}) \leftarrow (\text{direct}) \hat{\wedge} \#data$

---

## Chapter 6 Interrupts

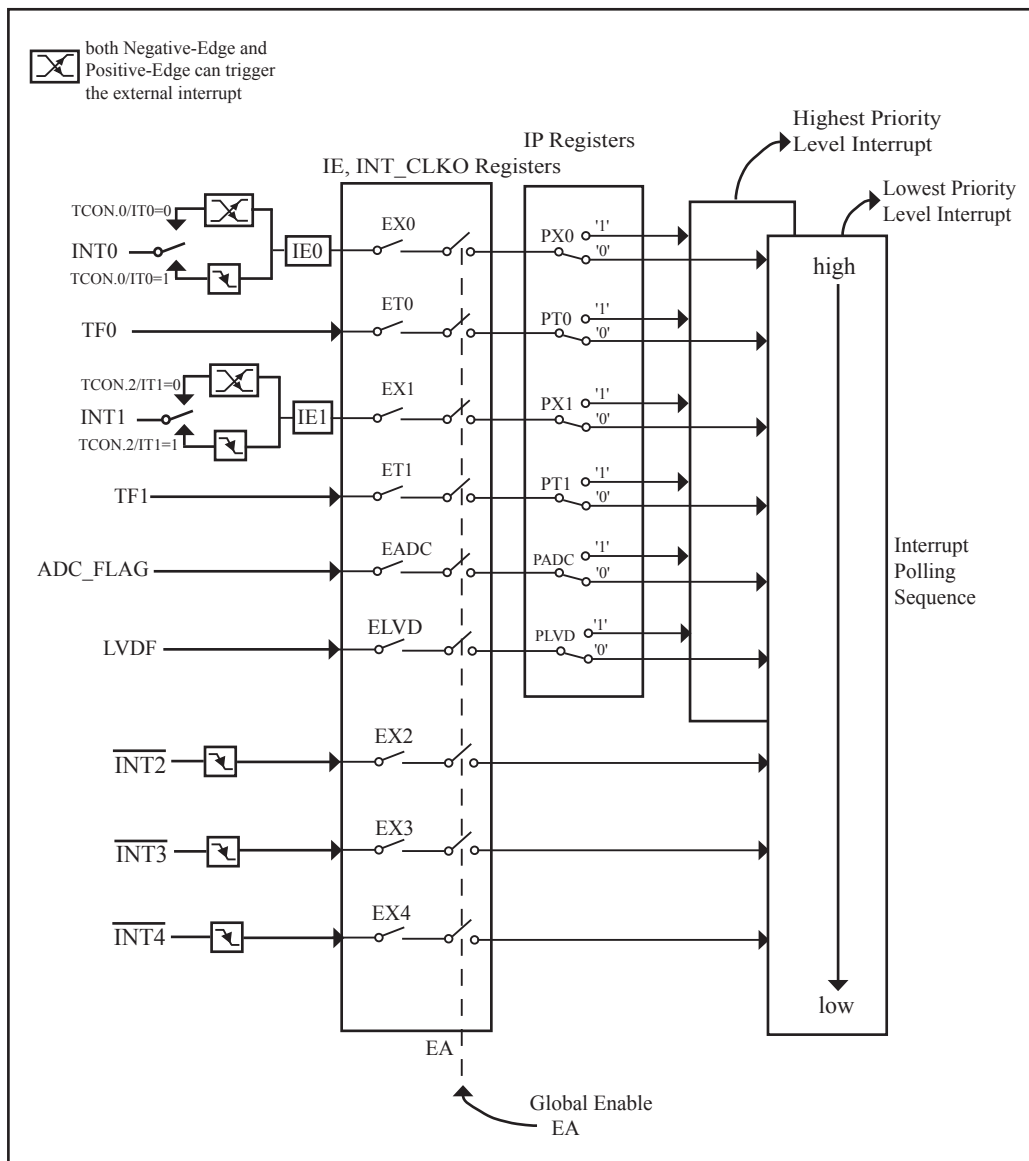
There are 9 interrupt vector addresses available in STC15F204EA series. Associating with each interrupt vector, the interrupt sources can be individually enabled or disabled by setting or clearing a bit in the registers IE, INT\_CLKO. These registers also contains a global disable bit(EA), which can 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 | Polling Sequence<br>(Priority within level) | Interrupt<br>Priority<br>Setting | Priority | Interrupt<br>Request Flag bit | Interrupt Enable<br>Control Bit |
|--------------------------------|----------------|---------------------------------------------|----------------------------------|----------|-------------------------------|---------------------------------|
| INT0<br>(External interrupt 0) | 0003H          | 0 (highest)                                 | PX0                              | 0/1      | IE0                           | EX0/EA                          |
| Timer 0                        | 000BH          | 1                                           | PT0                              | 0/1      | TF0                           | ET0/EA                          |
| INT1<br>(External interrupt 1) | 0013H          | 2                                           | PX1                              | 0/1      | IE1                           | EX1/EA                          |
| Timer1                         | 001BH          | 3                                           | PT1                              | 0/1      | TF1                           | ET1/EA                          |
| No S1(UART1)                   | 0023B          | 4                                           |                                  |          |                               |                                 |
| ADC                            | 002BH          | 5                                           | PADC                             | 0/1      | ADC_FLAG                      | EADC/EA                         |
| LVD                            | 0033H          | 6                                           | PLVD                             | 0/1      | LVDF                          | ELVD/EA                         |
| No PCA                         | 003BH          | 7                                           |                                  |          |                               |                                 |
| No S2(UART2)                   | 0043H          | 8                                           |                                  |          |                               |                                 |
| No SPI                         | 004BH          | 9                                           |                                  |          |                               |                                 |
| <u>INT2</u>                    | 0053H          | 10                                          |                                  | 0        |                               | EX2/EA                          |
| <u>INT3</u>                    | 005BH          | 11                                          |                                  | 0        |                               | EX3/EA                          |
| BRT_INT                        | 0063H          | 12                                          |                                  |          |                               |                                 |
| -                              | 006BH          | 13                                          |                                  |          |                               |                                 |
| System Reserved                | 0073H          | 14                                          |                                  |          |                               |                                 |
| System Reserved                | 007BH          | 15                                          |                                  |          |                               |                                 |
| <u>INT4</u>                    | 0083H          | 16                                          |                                  | 0        |                               | EX4/EA                          |

## 6.1 Interrupt Structure



STC15F204EA series 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  $\overline{\text{INT2}} \sim \overline{\text{INT4}}$  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                                                          |
|--------------------------------|-------------------------------------------------------------------------|
| INT0<br>(External interrupt 0) | (IT0 = 1): = Negative-Edge (IT0 = 0): = Positive-Edge and Negative-Edge |
| Timer 0                        | Timer0 overflow                                                         |
| INT1<br>(External interrupt 1) | (IT1 = 1): = Negative-Edge (IT1 = 0): = Positive-Edge and Negative-Edge |
| Timer1                         | Timer1 overflow                                                         |
| LVD                            | Power drops under LVD-setting level                                     |
| $\overline{\text{INT2}}$       | Negative-Edge                                                           |
| $\overline{\text{INT3}}$       | Negative-Edge                                                           |
| $\overline{\text{INT4}}$       | Negative-Edge                                                           |

## 6.2 Interrupt Register

| Symbol    | Description                                         | Address | Bit Address and Symbol |        |        |          |           |      |        |        | Value after Power-on or Reset |
|-----------|-----------------------------------------------------|---------|------------------------|--------|--------|----------|-----------|------|--------|--------|-------------------------------|
|           |                                                     |         | MSB                    |        |        |          | LSB       |      |        |        |                               |
| 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   | POF      | GF1       | GF0  | PD     | IDL    | xx11 0000B                    |
| 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 Register

| SFR Name | SFR Address | bit  | B7 | B6   | B5   | B4 | B3  | B2  | B1  | B0  |
|----------|-------------|------|----|------|------|----|-----|-----|-----|-----|
| IE       | A8H         | name | 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 voltage 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** (Address: B8H)

|       |      |      |   |       |     |     |     |
|-------|------|------|---|-------|-----|-----|-----|
| (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** (Address: 88H)

|       |     |     |     |       |     |     |     |
|-------|-----|-----|-----|-------|-----|-----|-----|
| (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   | Intenrupt 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   | Intenrupt 0 Type control bit. Set/ cleared by software to specify falling edge/low level triggered external interrupts. |

**PCON register** (Power Control Register)

| SFR name | Address | bit  | B7 | B6 | B5   | B4  | B3  | B2  | B1 | B0  |
|----------|---------|------|----|----|------|-----|-----|-----|----|-----|
| PCON     | 87H     | name | -  | -  | LVDF | POF | GF1 | GF0 | 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).

POF : Power-On flag. It is set by power-off-on action and can only cleared by software.

GF1 : General-purposed flag 1

GF0 : General-purposed flag 0

PD : Power-Down bit.

IDL : Idle mode bit.

---

**INT\_CLKO register**

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

EX4 : External interrupt 4 enable bit

0 : = Disable  $\overline{\text{INT4}}$  interrupt

1 : = Enable  $\overline{\text{INT4}}$  interrupt.

Only Negative-Edge from  $\overline{\text{INT4}}$  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  $\overline{\text{INT4}}$  can trigger interrupt as well as wakes up CPU from power-down mode.

EX3 : External interrupt 3 enable bit

0 : = Disable  $\overline{\text{INT3}}$  interrupt

1 : = Enable  $\overline{\text{INT3}}$  interrupt.

Only Negative-Edge from  $\overline{\text{INT3}}$  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  $\overline{\text{INT3}}$  can trigger interrupt as well as wakes up CPU from power-down mode.

EX2 : External interrupt 2 enable bit

0 := Disable  $\overline{\text{INT2}}$  interrupt

1 := Enable  $\overline{\text{INT2}}$  interrupt.

Only Negative-Edge from  $\overline{\text{INT2}}$  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  $\overline{\text{INT2}}$  can trigger interrupt as well as wakes up CPU from power-down mode.

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.

**ADC\_CONTR: AD Control register**

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

ADC\_POWER(ADC\_CONTR.7) : When clear, shut down the power of ADC block. When set, turn on the power of ADC block.

ADC\_FLAG(ADC\_CONTR.4) : ADC interrupt flag.

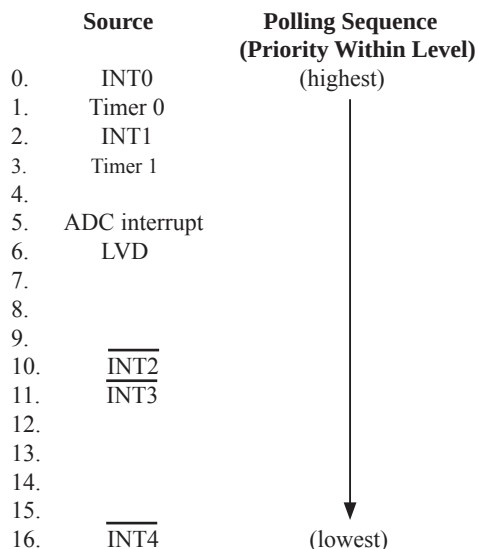
ADC\_STRAT : ADC start bit, which enable ADC conversion. It will automatically cleared by the device after the device has finished the conversion.

---

## 6.3 Interrupt Priorities

All interrupt sources, except  $\overline{\text{INT2}}$ ,  $\overline{\text{INT3}}$  and  $\overline{\text{INT4}}$ , 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-priority 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:



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

If programming in C language (Keil C), polling sequence is the interrupt number, for example:

```
void Int0_Routine(void)    interrupt 0;
void Timer0_Routine(void)  interrupt 1;
void Int1_Routine(void)    interrupt 2;
void Timer1_Routine(void)  interrupt 3;
void ADC_Routine(void)     interrupt 5;
void LVD_Routine(void)     interrupt 6;
void Int2_Routine(void)    interrupt 10;
void Int3_Routine(void)    interrupt 11;
void Int4_Routine(void)    interrupt 16;
```

---

## 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 below.

---

| Source               | Vector Address |
|----------------------|----------------|
| External Interrupt 0 | 0003H          |
| Timer 0              | 000BH          |
| External Interrupt 1 | 0013H          |
| Timer 1              | 001BH          |
| /                    | 0023H          |
| ADC interrupt        | 002BH          |
| LVD                  | 0033H          |
| /                    | 003BH          |
| /                    | 0043H          |
| /                    | 004BH          |
| External Interrupt 2 | 0053H          |
| External Interrupt 3 | 005BH          |
| /                    | 0063H          |
| /                    | 006BH          |
| /                    | 0073H          |
| /                    | 007BH          |
| 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  $\overline{\text{INT2}} \sim \overline{\text{INT4}}$  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.

The next texts list some demo procedures about how external interrupts operate.

External interrupt 0 (INT0) Demo program (written in C language):

```
/*-----*/
/* --- STC MCU International Limited -----*/
/* --- STC 15 Series MCU Ext0(Rising edge/Falling edge) Demo -----*/
/* 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"

bit FLAG;                                //1:rising edge int 0:falling edge int

//External interrupt0 service routine
void exint0() interrupt 0                 //interrupt 0 (location at 0003H)
{
    FLAG = INT0;                         //read INT0(P3.2) port status, INT0=0(Falling); INT0=1(Rising)
}

void main()
{
    IT0 = 0;                             //set INT0 int type (1:Falling only 0:Rising & Falling)
    EX0 = 1;                             //enable INT0 interrupt
    EA = 1;                              //open global interrupt switch
    while (1);
}
```

---

External interrupt 0 (INT0) Demo program (written in Assembly language) :

```
;/*-----*/
;/* --- STC MCU International Limited -----*/
;/* --- STC 15 Series MCU Ext0(Rising edge/Falling edge) Demo -----*/
;/* If you want to use the program or the program referenced in the -----*/
;/* article, please specify in which data and procedures from STC -----*/
;/*-----*/

                FLAG    BIT    20H.0                ;1:rising edge int 0:falling edge int

;-----
;interrupt vector table

                ORG      0000H
                LJMP     MAIN

                ORG      0003H                ;interrupt 0 (location at 0003H)
                LJMP     EXINT0

;-----

MAIN:           ORG      0100H
                MOV      SP,    #7FH                ;initial SP
                CLR      IT0                ;set INT0 int type (1:Falling only 0:Rising & Falling)
                SETB     EX0                ;enable INT0 interrupt
                SETB     EA                ;open global interrupt switch
                SJMP     $

;-----
;External interrupt0 service routine

EXINT0:
                PUSH     PSW
                MOV      C,      INT0                ;read INT0(P3.2) port status
                MOV      FLAG,    C                ;INT0=0(Falling); INT0=1(Rising)
                POP      PSW
                RETI

;-----

                END
```

---

External interrupt 1 (INT1) Demo program (written in C language) :

```
/*-----*/
/* --- STC MCU International Limited -----*/
/* --- STC 15 Series MCU Ext1(Rising edge/Falling edge) Demo -----*/
/* 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"

bit FLAG;                                //1:rising edge int 0:falling edge int

//External interrupt1 service routine
void exint1() interrupt 2                 //interrupt 2 (location at 0013H)
{
    FLAG = INT1;                         //read INT1(P3.3) port status, INT1=0(Falling); INT1=1(Rising)
}

void main()
{
    IT1 = 0;                             //set INT1 int type (1:Falling only 0:Rising & Falling)
    EX1 = 1;                             //enable INT1 interrupt
    EA = 1;                              //open global interrupt switch

    while (1);
}
```

---

External interrupt 1 (INT1) Demo program (written in Assembly language) :

```
;/*-----*/
;/* --- STC MCU International Limited -----*/
;/* --- STC 15 Series MCU Ext1(Rising edge/Falling edge) Demo -----*/
;/* If you want to use the program or the program referenced in the -----*/
;/* article, please specify in which data and procedures from STC -----*/
;/*-----*/

                FLAG    BIT    20H.0                ;1:rising edge int 0:falling edge int

;-----
;interrupt vector table

                ORG      0000H
                LJMP     MAIN

                ORG      0013H                ;interrupt 2 (location at 0013H)
                LJMP     EXINT1

;-----

                ORG      0100H
MAIN:
                MOV      SP,      #7FH                ;initial SP
                CLR      IT1                ;set INT1 int type (1:Falling only 0:Rising & Falling)
                SETB     EX1                ;enable INT1 interrupt
                SETB     EA                ;open global interrupt switch
                SJMP     $

;-----
;External interrupt1 service routine

EXINT1:
                PUSH     PSW
                MOV      C,      INT1                ;read INT1(P3.3) port status
                MOV      FLAG,    C                ;INT1=0(Falling); INT1=1(Rising)
                POP      PSW
                RETI

;-----

                END
```

---

External interrupt 2 ( $\overline{\text{INT2}}$ ) Demo program (written in C language) :

```
/*-----*/
/* --- STC MCU International Limited -----*/
/* --- STC 15 Series MCU Ext2(Falling edge) Demo -----*/
/* 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"

sfr INT_CLKO = 0x8f;          //- EX4 EX3 EX2 - - T1CLKO T0CLKO

//External interrupt2 service routine
void exint2() interrupt 10    //interrupt 10 (location at 0053H)
{
}

void main()
{
    INT_CLKO |= 0x10;        //(EX2 = 1)enable  $\overline{\text{INT2}}$  interrupt
    EA = 1;                  //open global interrupt switch

    while (1);
}
```

---

External interrupt 2 ( $\overline{\text{INT2}}$ ) Demo program (written in Assembly language) :

```
;/*-----*/
;/* --- STC MCU International Limited -----*/
;/* --- STC 15 Series MCU Ext2(Falling edge) Demo -----*/
;/* If you want to use the program or the program referenced in the ---*/
;/* article, please specify in which data and procedures from STC ---*/
;/*-----*/

INT_CLKO DATA 08FH                ; - EX4 EX3 EX2 - - T1CLKO T0CLKO

;-----
;interrupt vector table

    ORG 0000H
    LJMP MAIN

    ORG 0053H                      ;interrupt 10 (location at 0053H)
    LJMP EXINT2

;-----

    ORG 0100H
MAIN:  MOV SP, #7FH                ;initial SP
       ORL INT_CLKO, #10H         ;(EX2 = 1)enable  $\overline{\text{INT2}}$  interrupt
       SETB EA                    ;open global interrupt switch
       SJMP $

;-----
;External interrupt2 service routine

EXINT2:
    RETI

;-----

    END
```

---

External interrupt 3 ( $\overline{\text{INT3}}$ ) Demo program (written in C language) :

```
/*-----*/
/* --- STC MCU International Limited -----*/
/* --- STC 15 Series MCU Ext3(Falling edge) Demo -----*/
/* 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"

sfr INT_CLKO = 0x8f;          //- EX4 EX3 EX2 - - T1CLKO T0CLKO

//External interrupt3 service routine
void exint3() interrupt 11    //interrupt 11 (location at 005BH)
{
}

void main()
{
    INT_CLKO |= 0x20;        //(EX3 = 1)enable  $\overline{\text{INT3}}$  interrupt
    EA = 1;                  //open global interrupt switch

    while (1);
}
```

---

External interrupt 3 ( $\overline{\text{INT3}}$ ) Demo program (written in Assembly language) :

```
;/*-----*/
;/* --- STC MCU International Limited -----*/
;/* --- STC 15 Series MCU Ext3(Falling edge) Demo -----*/
;/* If you want to use the program or the program referenced in the ---*/
;/* article, please specify in which data and procedures from STC ---*/
;/*-----*/

INT_CLKO DATA 08FH ; - EX4 EX3 EX2 - - T1CLKO T0CLKO

;-----
;interrupt vector table

ORG 0000H
LJMP MAIN

ORG 005BH ;interrupt 11 (location at 005BH)
LJMP EXINT3

;-----

ORG 0100H
MAIN:
MOV SP, #7FH ;initial SP
ORL INT_CLKO, #20H ;(EX3 = 1)enable  $\overline{\text{INT3}}$  interrupt
SETB EA ;open global interrupt switch
SJMP $

;-----
;External interrupt 3 service routine

EXINT3:
RETI

;-----

END
```

---

External interrupt 4 ( $\overline{\text{INT4}}$ ) Demo program (written in C language) :

```
/*-----*/
/* --- STC MCU International Limited -----*/
/* --- STC 15 Series MCU Ext4(Falling edge) Demo -----*/
/* 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"

sfr INT_CLKO = 0x8f;          //- EX4 EX3 EX2 -- T1CLKO T0CLKO

//External interrupt4 service routine
void exint4() interrupt 16    //interrupt 16 (location at 0083H)
{
}

void main()
{
    INT_CLKO |= 0x40;        //(EX4 = 1)enable  $\overline{\text{INT4}}$  interrupt
    EA = 1;                  //open global interrupt switch

    while (1);
}
```

---

External interrupt 4 (INT4) Demo program (written in Assembly language) :

```
;/*-----*/
;/* --- STC MCU International Limited -----*/
;/* --- STC 15 Series MCU Ext4(Falling edge) Demo -----*/
;/* If you want to use the program or the program referenced in the --*/
;/* article, please specify in which data and procedures from STC ---*/
;/*-----*/

INT_CLKO DATA 08FH ; - EX4 EX3 EX2 - - T1CLKO T0CLKO

;-----
;interrupt vector table

ORG 0000H
LJMP MAIN

ORG 0083H ;interrupt 16 (location at 0083H)
LJMP EXINT4

;-----

ORG 0100H
MAIN:
MOV SP, #7FH ;initial SP
ORL INT_CLKO, #40H ;(EX4 = 1)enable INT4 interrupt
SETB EA ;open global interrupt switch
SJMP $

;-----
;External interrupt4 service routine

EXINT4:
RETI

;-----

END
```

## 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 1-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 |       |     |     |      |     |        |        | Value after Power-on or Reset |
|----------|-----------------------------------------------------|---------|------------------------|-------|-----|-----|------|-----|--------|--------|-------------------------------|
|          |                                                     |         | MSB                    |       |     |     | LSB  |     |        |        |                               |
| TCON     | Timer Control                                       | 88H     | TF1                    | TR1   | TF0 | TR0 | IE1  | IT1 | IE0    | IT0    | 0000 0000B                    |
| TMOD     | Timer Mode                                          | 89H     | GATE                   | C/T   | M1  | M0  | GATE | C/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** (Address: 88H)

| (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** (Address: 89H)

| (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<br>TH0 is an 8-bit timer only controlled by Timer 1 control bits. |
| 1  | 1  | (Timer 1) Timer/Counter 1 stopped                                                                                                                         |

---

**AUXR register** (Address:8EH)

LSB

| SFR name | Address | bit  | B7    | B6    | B5 | B4 | B3 | B2 | B1 | B0 |
|----------|---------|------|-------|-------|----|----|----|----|----|----|
| AUXR     | 8EH     | 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**

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

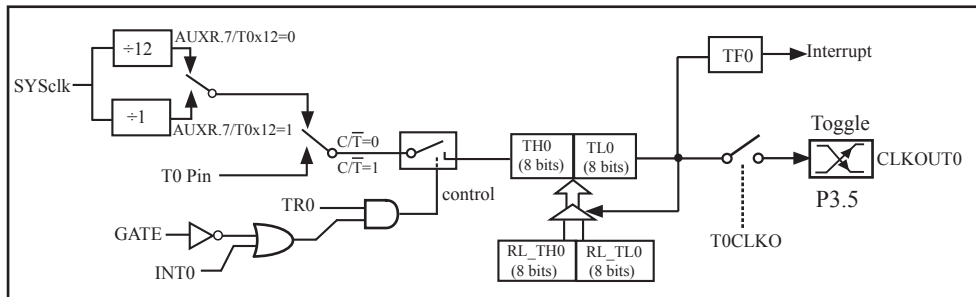
T1CLKO : When set, P3.4 is enabled to be the clock output of Timer 1. The clock rate is Timer 1 overflow rate divided by 2.

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 address of RL\_TL0/RL\_TH0 are homogeneous to TL0/TH0.

While the Timer 0 is configured to operate under Mode 0 (TMOD[1:0]/[M1, M0] = 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].

STC15F204EA series 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 :

$$\begin{aligned} & \frac{(\text{SYSclock}/2)}{(256 - \text{TH0})}, & \text{when } \text{T0x12}=1 \\ \text{or } & \frac{(\text{SYSclock}/2/12)}{(256 - \text{TH0})}, & \text{when } \text{T0x12}=0 \end{aligned}$$

---

The following program is an C language code that demonstrates Timer 0 in 16-bit auto-reload timer mode.

```
/*-----*/
/* --- STC MCU International Limited -----*/
/* --- STC 15 Series 16-bit auto-reload Timer Demo -----*/
/* 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 1843200L
#define MODE1T //Timer clock mode, comment this line is 12T mode, uncomment is 1T mode

#ifndef 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 = P0^0; //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
}

```

The following program is the same as the above program except in assembly language.

```

; /* ----- */
; /* --- STC MCU International Limited ----- */
; /* --- STC 15 Series 16-bit auto-reload Timer Demo ----- */
; /* 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

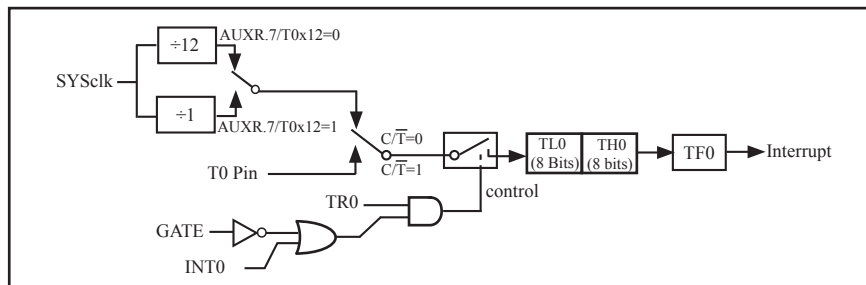
```

---

## 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.



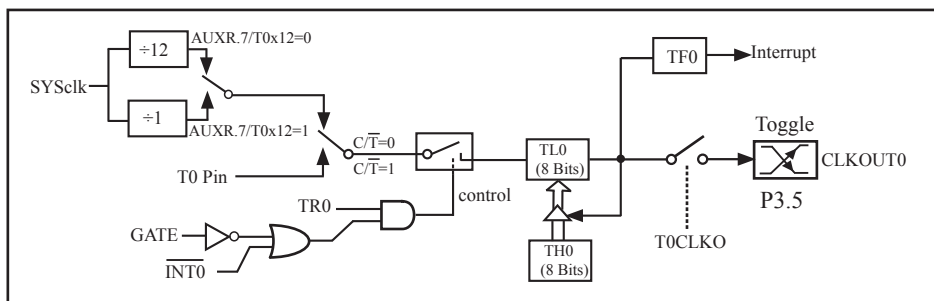
Timer/Counter 0 Mode 1 : 16-Bit Timer/Counter

## Mode 2

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

STC15F204EA series 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 :

$$\begin{aligned} & \frac{(\text{SYSclk}/2)}{(256 - \text{TH0})}, & \text{when } \text{T0x12}=1 \\ \text{or } & \frac{(\text{SYSclk}/2/12)}{(256 - \text{TH0})}, & \text{when } \text{T0x12}=0 \end{aligned}$$



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 Tmter 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 */
}

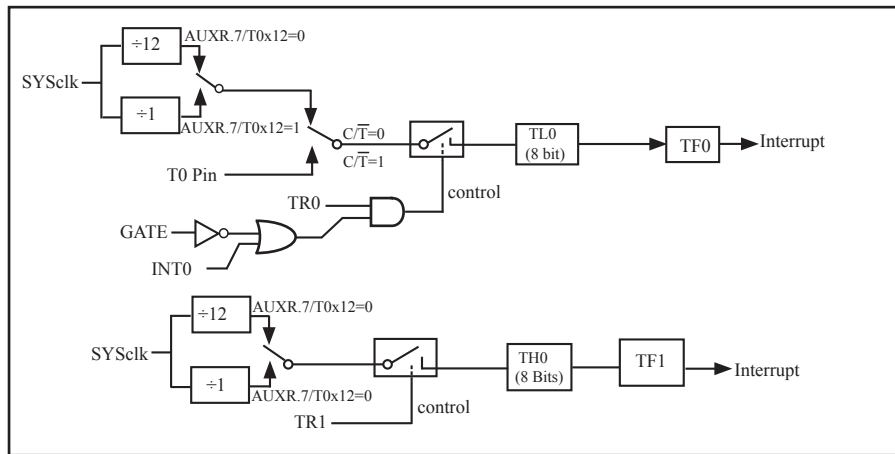
void T0ISR(void) interrupt 1
{
    portbit = !portbit;       /*toggle port bit P1.0 */
}
```

---

### 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:  $\overline{C/T}$ ,  $GATE$ ,  $TR0$ ,  $\overline{INT0}$  and  $TF0$ .  $TH0$  is locked into a timer function (counting machine cycles) and takes over the use of  $TR1$  from Timer 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.

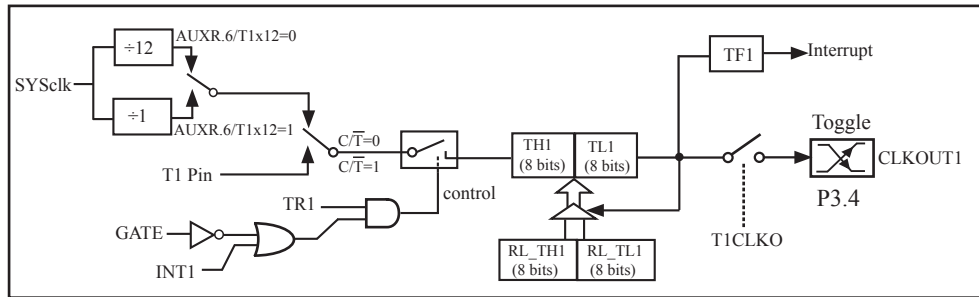


Timer/Counter 0 Mode 3: Two 8-Bit Counters

## 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 address of RL\_TL1/RL\_TH1 are homogeneous to TL1/TH1.

While the Timer 1 is configured to operate under Mode 0 (TMOD[5:4]/[M1, M0] = 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].

STC15F204EA series 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 :

$$\begin{aligned} & \text{when } T0x12=1 \\ & \text{or } (\text{SYSClk}/2) / (256 - TH0), \\ & \text{when } T0x12=0 \\ & (\text{SYSClk}/2/12) / (256 - TH0), \end{aligned}$$

---

The following program is an assembly language code that demonstrates Timer 1 in 16-bit auto-reload timer mode.

```
/*-----*/
/* --- STC MCU International Limited -----*/
/* --- STC 15 Series 16-bit auto-reload Timer Demo -----*/
/* 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 1843200L
#define MODE1T //Timer clock mode, comment this line is 12T mode, uncomment is 1T mode

#ifndef 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 = P0^0; //work LED, flash once per second

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

//-----

/* Timer1 interrupt routine */
void tm1_isr() interrupt 3 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 = 0x40;           //timer1 work in 1T mode
#endif

    TMOD = 0x00;           //set timer1 as mode0 (16-bit auto-reload)
    TL1 = T1MS;            //initial timer1 low byte
    TH1 = T1MS >> 8;       //initial timer1 high byte
    TR1 = 1;               //timer1 start running
    ET1 = 1;               //enable timer1 interrupt
    EA = 1;                //open global interrupt switch
    count = 0;             //initial counter

    while (1);             //loop
}

```

The following program is as the same as the above program except in assembly language.

```

; /*-----*/
; /* --- STC MCU International Limited -----*/
; /* --- STC 15 Series 16-bit auto-reload Timer Demo -----*/
; /* 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     001BH
        LJMP    TM1_ISR
;-----

; /* main program */
MAIN:
#ifdef MODE1T
        MOV     AUXR,  #40H                ;timer1 work in 1T mode
#endif
        MOV     TMOD,  #00H                ;set timer1 as mode0 (16-bit auto-reload)
        MOV     TL1,   #LOW T1MS           ;initial timer1 low byte
        MOV     TH1,   #HIGH T1MS          ;initial timer1 high byte
        SETB    TR1                        ;timer1 start running
        SETB    ET1                        ;enable timer1 interrupt
        SETB    EA                        ;open global interrupt switch
        CLR     A
        MOV     COUNT,  A
        MOV     COUNT+1, A                ;initial counter
        SJMP    $
;-----

; /* Timer1 interrupt routine */
TM1_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

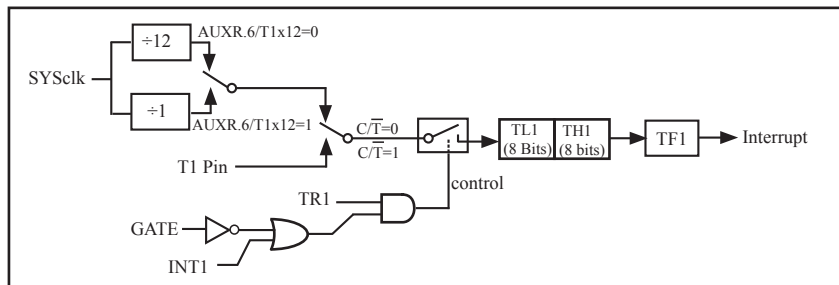
```

---

## 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.) TR1 is a control bit in the Special Function Register TCON. GATE is in TMOD.

The 16-Bit register consists of all 8 bits of TH1 and the lower 8 bits of TL1. Setting the run flag (TR1) does not clear the registers.



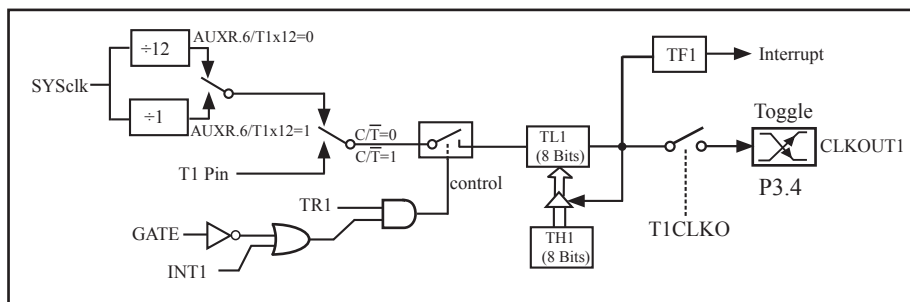
Timer/Counter 1 Mode 1 : 16-Bit Counter

## 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.

STC15F204EA series 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 :

$$\begin{aligned} & \text{when } T1x12=1, \\ & \text{or } \frac{(\text{SYSclk}/2)}{(256 - \text{TH1})}, \\ & \text{when } T1x12=0, \end{aligned}$$



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

---

## 7.3 Generic Programmable Clock Output

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

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

| SFR Name | SFR Address | bit  | B7      | B6 | B5 | B4 | B3      | B2 | B1 | B0 |
|----------|-------------|------|---------|----|----|----|---------|----|----|----|
| IRC_CLKO | BBH         | name | 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  $\text{SYSclk}/2$  or  $\text{SYSclk}$ .

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 :

$$\begin{aligned} & \text{or } \frac{(\text{SYSclk}/2)}{(256 - \text{TH0})}, & \text{when T0x12=1} \\ & \frac{(\text{SYSclk}/2/12)}{(256 - \text{TH0})}, & \text{when T0x12=0} \end{aligned}$$

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 :

$$\begin{aligned} & \text{or } \frac{(\text{SYSclk}/2)}{(256 - \text{TH1})}, & \text{when T1x12=1} \\ & \frac{(\text{SYSclk}/2/12)}{(256 - \text{TH1})}, & \text{when T1x12=0} \end{aligned}$$

---

The following program is an C language code that demonstrates Internal RC oscillator Clock Output function.

```
/*-----*/
/* --- STC MCU International Limited -----*/
/* --- STC 15 Series MCU IRC clock output Demo -----*/
/* If you want to use the program or the program referenced in the */
/* article, please specify in which data and procedures from STC */
/*-----*/

sfr IRC_CLKO = 0xbb;          //EN_IRCO - - - DIVIRCO - - -

//-----

void main()
{
    IRC_CLKO = 0x80;          //1000,0000 P0.0 output clock signal which frequency is SYSclk
//    IRC_CLKO = 0x88;          //1000,1000 P0.0 output clock signal which frequency is SYSclk/2

    while (1);
}
```

---

The following program is an Assembly language code that demonstrates Internal RC oscillator Clock Output function.

```
;/*-----*/
;/* --- STC MCU International Limited -----*/
;/* --- STC 15 Series MCU IRC clock output Demo -----*/
;/* If you want to use the program or the program referenced in the */
;/* article, please specify in which data and procedures from STC */
;/*-----*/

IRC_CLKO DATA 0BBH ;EN_IRCO --- DIVIRCO ---

;-----
;interrupt vector table

ORG 0000H
LJMP MAIN

;-----

ORG 0100H
MAIN:
MOV SP,#7FH ;initial SP
MOV IRC_CLKO, #80H ;1000,0000 P0.0 output clock signal which frequency is SYSclk
; MOV IRC_CLKO,#88H ;1000,1000
; ;P0.0 output clock signal which frequency is SYSclk/2

SJMP $

;-----

END
```

---

The following program is an C language code that demonstrates Timer 0 as Programmable Clock Output function.

```
/*-----*/
/* --- STC MCU International Limited -----*/
/* --- STC 15 Series Programmable Clock Output Demo -----*/
/* 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 1843200L
// #define MODE1T           //Timer clock mode, comment this line is 12T mode, uncomment is 1T mode

#ifndef 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 INT_CLKO = 0x8f;           //External interrupt enable and clock output control register
sbit T0CLKO  = P3^5;           //timer0 clock output pin

//-----

/* main program */
void main()
{
#ifndef 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
    INT_CLKO = 0x01;           //enable timer0 clock output

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

---

The following program is an assembly language code that demonstrates Timer 0 as Programmable Clock Output function.

```
;/*-----*/
;/* --- STC MCU International Limited -----*/
;/* --- STC 15 Series Programmable Clock Output Demo -----*/
;/* 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

#ifndef 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
        INT_CLKO       DATA 08FH      ;External interrupt enable and clock output control register
        T0CLKO         BIT  P3.5       ;timer0 clock output pin

;-----
        ORG    0000H
        LJMP   MAIN

;-----
;/* main program */
MAIN:
#ifndef 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     INT_CLKO, #01H          ;enable timer0 clock output

        SJMP    $

;-----
        END
```

---

The following program is an C language code that demonstrates Timer 1 as Programmable Clock Output function.

```
/*-----*/
/* --- STC MCU International Limited -----*/
/* --- STC 15 Series Programmable Clock Output Demo -----*/
/* 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 1843200L
// #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 INT_CLKO = 0x8f; //External interrupt enable and clock output control register
sbit T1CLKO = P3^4; //timer1 clock output pin

//-----

/* main program */
void main()
{
#ifdef MODE1T
    AUXR = 0x40; //timer1 work in 1T mode
#endif
    TMOD = 0x00; //set timer1 as mode0 (16-bit auto-reload)
    TL1 = F38_4KHz; //initial timer1 low byte
    TH1 = F38_4KHz >> 8; //initial timer1 high byte
    TR1 = 1; //timer1 start running
    INT_CLKO = 0x02; //enable timer1 clock output

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

---

The following program is an assembly language code that demonstrates Timer 1 as Programmable Clock Output function.

```

; /*-----*/
; /* --- STC MCU International Limited -----*/
; /* --- STC 15 Series Programmable Clock Output Demo -----*/
; /* 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

#ifndef 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
INT_CLKO   DATA    08FH      ;External interrupt enable and clock output control register
T1CLKO     BIT      P3.4      ;timer1 clock output pin

;-----
                ORG     0000H
                LJMP    MAIN
;-----
; /* main program */
MAIN:
#ifndef MODE1T
    MOV     AUXR, #40H          ;timer1 work in 1T mode
#endif
    MOV     TMOD, #00H          ;set timer1 as mode0 (16-bit auto-reload)
    MOV     TL1,  #LOW F38_4KHz ;initial timer1 low byte
    MOV     TH1,  #HIGH F38_4KHz ;initial timer1 high byte
    SETB    TR1
    MOV     INT_CLKO, #02H      ;enable timer1 clock output

    SJMP    $

;-----

                END

```

---

## 7.4 Changes of STC15F204EA series Timers compared with standard 8051

The Timer 0 and Timer1 are almost the same to standard 80C51 MCU excepting the following changes.

### Timer0 and Timer1 Clock Sources

| SFR Name | SFR Address | bit  | B7    | B6    | B5 | B4 | B3 | B2 | B1 | B0 |
|----------|-------------|------|-------|-------|----|----|----|----|----|----|
| AUXR     | 97H         | name | 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.

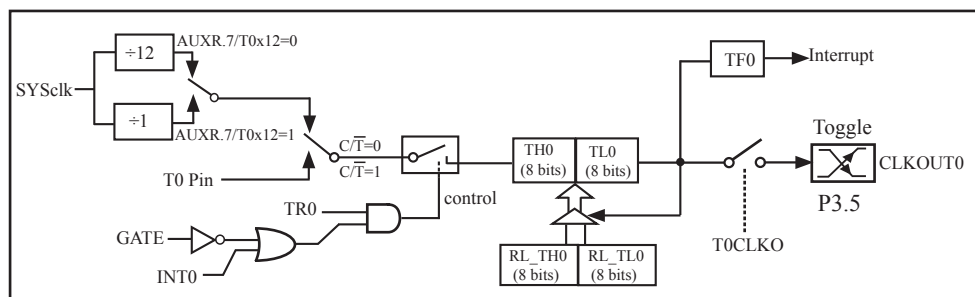
### Change MODE0 functionality

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 address 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]/[M1,M0] = 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.

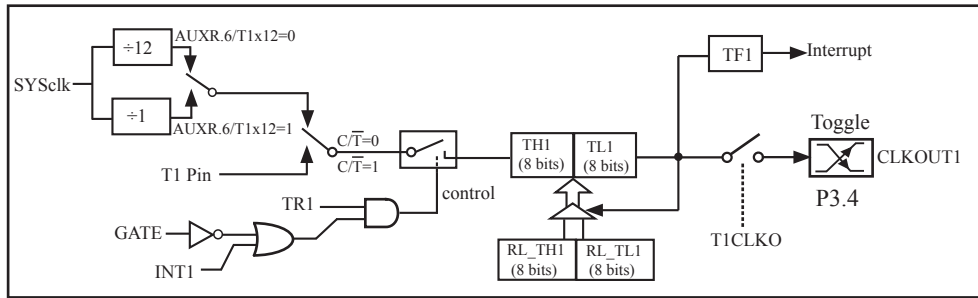
Under MODE0 operating, overflow of [TH0,TL0] will automatically reload value [RL\_TH0,RL\_TL0] onto [TH0,TL0].



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

While the Timer1 is configured to operate under MODE0 (TMOD[5:4]/[M1,M0] = 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].



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

---

## Chapter 8 Simulate Serial Port Program

### 8.1 Programs using Timer 0 to realize Simulate Serial Port

#### ---Timer 0 in 16-bit Auto-Reload Mode

There are two procedures using Timer 0 to realize simulate serial port, one written in C language and the other written in Assembly language. Timer 0 in the following two programs both operate in 16-bit auto-reload mode.

#### C language code listing:

```
/*-----*/
/* --- STC MCU International Limited -----*/
/* --- STC 15 Series I/O simulate serial port -----*/
/* 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 baudrate const
//BAUD = 256 - FOSC/3/BAUDRATE/M (1T:M=1; 12T:M=12)
//NOTE: (FOSC/3/BAUDRATE) must be greater than 98, (RECOMMEND GREATER THAN 110)

#define BAUD 0xF400 // 1200bps @ 11.0592MHz
#define BAUD 0xFA00 // 2400bps @ 11.0592MHz
#define BAUD 0xFD00 // 4800bps @ 11.0592MHz
#define BAUD 0xFE80 // 9600bps @ 11.0592MHz
#define BAUD 0xFF40 // 19200bps @ 11.0592MHz
#define BAUD 0xFFA0 // 38400bps @ 11.0592MHz

#define BAUD 0xEC00 // 1200bps @ 18.432MHz
#define BAUD 0xF600 // 2400bps @ 18.432MHz
#define BAUD 0xFB00 // 4800bps @ 18.432MHz
#define BAUD 0xFD80 // 9600bps @ 18.432MHz
#define BAUD 0xFEC0 // 19200bps @ 18.432MHz
#define BAUD 0xFF60 // 38400bps @ 18.432MHz

#define BAUD 0xE800 // 1200bps @ 22.1184MHz
#define BAUD 0xF400 // 2400bps @ 22.1184MHz
#define BAUD 0xFA00 // 4800bps @ 22.1184MHz
#define BAUD 0xFD00 // 9600bps @ 22.1184MHz
#define BAUD 0xFE80 // 19200bps @ 22.1184MHz
#define BAUD 0xFF40 // 38400bps @ 22.1184MHz
#define BAUD 0xFF80 // 57600bps @ 22.1184MHz
```

---

```

sfr AUXR = 0x8E;
sbit RXB = P3^0;                                //define UART TX/RX port
sbit TXB = P3^1;

typedef bit BOOL;
typedef unsigned char BYTE;
typedef unsigned int WORD;

BYTE TBUF,RBUF;
BYTE TDAT,RDAT;
BYTE TCNT,RCNT;
BYTE TBIT,RBIT;
BOOL TING,RING;
BOOL TEND,REND;

void UART_INIT();

BYTE t, r;
BYTE buf[16];

void main()
{
    TMOD = 0x00;                                //timer0 in 16-bit auto reload mode
    AUXR = 0x80;                                //timer0 working at 1T mode
    TL0 = BAUD;
    TH0 = BAUD>>8;                             //initial timer0 and set reload value
    TR0 = 1;                                    //timer0 start running
    ET0 = 1;                                    //enable timer0 interrupt
    PT0 = 1;                                    //improve timer0 interrupt priority
    EA = 1;                                     //open global interrupt switch

    UART_INIT();

    while (1)
    {
        //user's function
        if (REND)
        {
            REND = 0;
            buf[r++ & 0x0f] = RBUF;
        }
        if (TEND)
        {
            if (t != r)
            {
                TEND = 0;
                TBUF = buf[t++ & 0x0f];
                TING = 1;
            }
        }
    }
}

```

---

---

```

//-----
//Timer interrupt routine for UART

void tm0() interrupt 1 using 1
{
    if (RING)
    {
        if (--RCNT == 0)
        {
            RCNT = 3;           //reset send baudrate counter
            if (--RBIT == 0)
            {
                RBUF = RDAT;     //save the data to RBUF
                RING = 0;        //stop receive
                REND = 1;        //set receive completed flag
            }
            else
            {
                RDAT >>= 1;
                if (RXB) RDAT |= 0x80;    //shift RX data to RX buffer
            }
        }
    }
    else if (!RXB)
    {
        RING = 1;               //set start receive flag
        RCNT = 4;               //initial receive baudrate counter
        RBIT = 9;               //initial receive bit number (8 data bits + 1 stop bit)
    }

    if (--TCNT == 0)
    {
        TCNT = 3;               //reset send baudrate counter
        if (TING)               //judge whether sending
        {
            if (TBIT == 0)
            {
                TXB = 0;         //send start bit
                TDAT = TBUF;     //load data from TBUF to TDAT
                TBIT = 9;        //initial send bit number (8 data bits + 1 stop bit)
            }
        }
    }
}

```

---

---

```

        else
        {
            TDAT >>= 1;    //shift data to CY
            if (--TBIT == 0)
            {
                TXB = 1;
                TING = 0;    //stop send
                TEND = 1;    //set send completed flag
            }
            else
            {
                TXB = CY;    //write CY to TX port
            }
        }
    }
}

//-----
//initial UART module variable

void UART_INIT()
{
    TING = 0;
    RING = 0;
    TEND = 1;
    REND = 0;
    TCNT = 0;
    RCNT = 0;
}

```

---

## Assembly language code listing:

```
/*-----*/
/* --- STC MCU International Limited -----*/
/* --- STC 15 Series I/O simulate serial port -----*/
/* If you want to use the program or the program referenced in the */
/* article, please specify in which data and procedures from STC */
/*-----*/

;-----
;define baudrate const
;BAUD = 65536 - FOSC/3/BAUDRATE/M (1T:M=1; 12T:M=12)
;NOTE: (FOSC/3/BAUDRATE) must be greater then 75, (RECOMMEND GREATER THEN 100)

;BAUD EQU 0F400H ; 1200bps @ 11.0592MHz
;BAUD EQU 0FA00H ; 2400bps @ 11.0592MHz
;BAUD EQU 0FD00H ; 4800bps @ 11.0592MHz
;BAUD EQU 0FE80H ; 9600bps @ 11.0592MHz
;BAUD EQU 0FF40H ; 19200bps @ 11.0592MHz
;BAUD EQU 0FFA0H ; 38400bps @ 11.0592MHz
;BAUD EQU 0FFC0H ; 57600bps @ 11.0592MHz

;BAUD EQU 0EC00H ; 1200bps @ 18.432MHz
;BAUD EQU 0F600H ; 2400bps @ 18.432MHz
;BAUD EQU 0FB00H ; 4800bps @ 18.432MHz
;BAUD EQU 0FD80H ; 9600bps @ 18.432MHz
;BAUD EQU 0FEC0H ; 19200bps @ 18.432MHz
;BAUD EQU 0FF60H ; 38400bps @ 18.432MHz
BAUD EQU 0FF95H ; 57600bps @ 18.432MHz

;BAUD EQU 0E800H ; 1200bps @ 22.1184MHz
;BAUD EQU 0F400H ; 2400bps @ 22.1184MHz
;BAUD EQU 0FA00H ; 4800bps @ 22.1184MHz
;BAUD EQU 0FD00H ; 9600bps @ 22.1184MHz
;BAUD EQU 0FE80H ; 19200bps @ 22.1184MHz
;BAUD EQU 0FF40H ; 38400bps @ 22.1184MHz
;BAUD EQU 0FF80H ; 57600bps @ 22.1184MHz
```

---

```

;-----
;define UART TX/RX port

RXB  BIT  P3.0
TXB  BIT  P3.1

;-----
;define SFR

AUXR  DATA  8EH

;-----
;define UART module variable

TBUF  DATA  08H      ;(R0) ready send data buffer  (USER WRITE ONLY)
RBUF  DATA  09H      ;(R1) received data buffer  (UAER READ ONLY)
TDAT  DATA  0AH      ;(R2) sending data buffer  (RESERVED FOR UART MODULE)
RDAT  DATA  0BH      ;(R3) receiving data buffer  (RESERVED FOR UART MODULE)
TCNT  DATA  0CH      ;(R4) send baudrate counter  (RESERVED FOR UART MODULE)
RCNT  DATA  0DH      ;(R5) receive baudrate counter (RESERVED FOR UART MODULE)
TBIT  DATA  0EH      ;(R6) send bit counter    (RESERVED FOR UART MODULE)
RBIT  DATA  0FH      ;(R7) receive bit counter   (RESERVED FOR UART MODULE)

TING  BIT  20H.0      ; sending flag  (USER WRITE "1" TO TRIGGER SEND DATA, CLEAR BY
MODULE)
RING  BIT  20H.1      ; receiving flag (RESERVED FOR UART MODULE)
TEND  BIT  20H.2      ; sent flag   (SET BY MODULE AND SHOULD USER CLEAR)
REND  BIT  20H.3      ; received flag (SET BY MODULE AND SHOULD USER CLEAR)

RPTR  DATA  21H      ;circular queue read pointer
WPTR  DATA  22H      ;circular queue write pointer
BUFFER DATA  23H      ;circular queue buffer (16 bytes)

;-----

    ORG  0000H
    LJMP  RESET

;-----
;Timer0 interrupt routine for UART

    ORG  000BH

    PUSH  ACC          ;4 save ACC
    PUSH  PSW          ;4 save PSW
    MOV   PSW, #08H    ;3 using register group 1
L_UARTSTART:
;-----

```

---

---

|            |      |        |        |                                                         |
|------------|------|--------|--------|---------------------------------------------------------|
|            | JB   | RING,  | L_RING | ;4 judge whether receiving                              |
|            | JB   | RXB,   | L_REND | ; check start signal                                    |
| L_RSTART:  |      |        |        |                                                         |
|            | SETB | RING   |        | ; set start receive flag                                |
|            | MOV  | R5,    | #4     | ; initial receive baudrate counter                      |
|            | MOV  | R7,    | #9     | ; initial receive bit number (8 data bits + 1 stop bit) |
|            | SJMP | L_REND |        | ; end this time slice                                   |
| L_RING:    |      |        |        |                                                         |
|            | DJNZ | R5,    | L_REND | ;4 judge whether sending                                |
|            | MOV  | R5,    | #3     | ;2 reset send baudrate counter                          |
| L_RBIT:    |      |        |        |                                                         |
|            | MOV  | C,     | RXB    | ;3 read RX port data                                    |
|            | MOV  | A,     | R3     | ;1 and shift it to RX buffer                            |
|            | RRC  | A      |        | ;1                                                      |
|            | MOV  | R3,    | A      | ;2                                                      |
|            | DJNZ | R7,    | L_REND | ;4 judge whether the data have receive completed        |
| L_RSTOP:   |      |        |        |                                                         |
|            | RLC  | A      |        | ; shift out stop bit                                    |
|            | MOV  | R1,    | A      | ; save the data to RBUF                                 |
|            | CLR  | RING   |        | ; stop receive                                          |
|            | SETB | REND   |        | ; set receive completed flag                            |
| L_REND:    |      |        |        |                                                         |
|            | ;    | -----  |        |                                                         |
| L_TING:    |      |        |        |                                                         |
|            | DJNZ | R4,    | L_TEND | ;4 check send baudrate counter                          |
|            | MOV  | R4,    | #3     | ;2 reset it                                             |
|            | JNB  | TING,  | L_TEND | ;4 judge whether sending                                |
|            | MOV  | A,     | R6     | ;1 detect the sent bits                                 |
|            | JNZ  | L_TBIT |        | ;3 "0" means start bit not sent                         |
| L_TSTART:  |      |        |        |                                                         |
|            | CLR  | TXB    |        | ; send start bit                                        |
|            | MOV  | TDAT,  | R0     | ; load data from TBUF to TDAT                           |
|            | MOV  | R6,    | #9     | ; initial send bit number (8 data bits + 1 stop bit)    |
|            | JMP  | L_TEND |        | ; end this time slice                                   |
| L_TBIT:    |      |        |        |                                                         |
|            | MOV  | A,     | R2     | ;1 read data in TDAT                                    |
|            | SETB | C      |        | ;1 shift in stop bit                                    |
|            | RRC  | A      |        | ;1 shift data to CY                                     |
|            | MOV  | R2,    | A      | ;2 update TDAT                                          |
|            | MOV  | TXB,   | C      | ;4 write CY to TX port                                  |
|            | DJNZ | R6,    | L_TEND | ;4 judge whether the data have send completed           |
| L_TSTOP:   |      |        |        |                                                         |
|            | CLR  | TING   |        | ; stop send                                             |
|            | SETB | TEND   |        | ; set send completed flag                               |
| L_TEND:    |      |        |        |                                                         |
|            | ;    | -----  |        |                                                         |
| L_UARTEND: |      |        |        |                                                         |
|            | POP  | PSW    |        | ;3 restore PSW                                          |
|            | POP  | ACC    |        | ;3 restore ACC                                          |
|            | RETI |        |        | ;4 (69)                                                 |

---

---

```

;-----
;initial UART module variable

UART_INIT:
    CLR    TING
    CLR    RING
    SETB   TEND
    CLR    REND
    CLR    A
    MOV    TCNT,  A
    MOV    RCNT,  A
    RET

;-----
;main program entry

RESET:
    MOV    R0,    #7FH                ;clear RAM
    CLR    A
    MOV    @R0,   A
    DJNZ   R0,    $-1
    MOV    SP,    #7FH                ;initial SP

;-----
;system initial
    MOV    TMOD,  #00H                ;timer0 in 16-bit auto reload mode
    MOV    AUXR,  #80H                ;timer0 working at 1T mode
    MOV    TL0,   #LOW BAUD           ;initial timer0 and
    MOV    TH0,   #HIGH BAUD         ;set reload value
    SETB   TR0                        ;tiemr0 start running
    SETB   ET0                        ;enable timer0 interrupt
    SETB   PT0                        ;improve timer0 interrupt priority
    SETB   EA                        ;open global interrupt switch
    LCALL  UART_INIT

;-----
MAIN:
    JNB    REND,  CHECKREND           ;if (REND)
    CLR    REND                        ;{
    MOV    A,     RPTR                ;    REND = 0;
    INC    RPTR                        ;    BUFFER[RPTR++ & 0xf] = RBUF;
    ANL    A,     #0FH                ;}
    ADD    A,     #BUFFER              ;
    MOV    R0,    A                    ;
    MOV    @R0,   RBUF                 ;

```

---

---

CHECKREND:

```
JNB    TEND,  MAIN          ;if (TEND)
MOV     A,    RPTR          ;{
XRL     A,    WPTR          ;    if (WPTR != REND)
JZ      MAIN          ;    {
CLR     TEND          ;                TEND = 0;
MOV     A,    WPTR          ;                TBUF = BUFFER[WPTR++ & 0xf];
INC     WPTR          ;                TING = 1;
ANL     A,    #0FH          ;    }
ADD     A,    #BUFFER      ;}
MOV     R0,    A          ;
MOV     TBUF,  @R0          ;
SETB    TING          ;
SJMP    MAIN
```

;-----

END

---

## 8.2 Programs using Timer 1 to realize Simulate Serial Port

### ---Timer 1 in 16-bit Auto-Reload Mode

There are two procedures using Timer 1 to realize simulate serial port, one written in C language and the other written in Assembly language. Timer 1 in the following two programs both operate in 16-bit auto-reload mode.

#### C language code listing:

```
/*-----*/
/* --- STC MCU International Limited -----*/
/* --- STC 15 Series I/O simulate serial port -----*/
/* 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 baudrate const
//BAUD = 256 - FOSC/3/BAUDRATE/M (1T:M=1; 12T:M=12)
//NOTE: (FOSC/3/BAUDRATE) must be greater than 98, (RECOMMEND GREATER THAN 110)

#define BAUD 0xF400 // 1200bps @ 11.0592MHz
#define BAUD 0xFA00 // 2400bps @ 11.0592MHz
#define BAUD 0xFD00 // 4800bps @ 11.0592MHz
#define BAUD 0xFE80 // 9600bps @ 11.0592MHz
#define BAUD 0xFF40 // 19200bps @ 11.0592MHz
#define BAUD 0xFFA0 // 38400bps @ 11.0592MHz

#define BAUD 0xEC00 // 1200bps @ 18.432MHz
#define BAUD 0xF600 // 2400bps @ 18.432MHz
#define BAUD 0xFB00 // 4800bps @ 18.432MHz
#define BAUD 0xFD80 // 9600bps @ 18.432MHz
#define BAUD 0xFEC0 // 19200bps @ 18.432MHz
#define BAUD 0xFF60 // 38400bps @ 18.432MHz

#define BAUD 0xE800 // 1200bps @ 22.1184MHz
#define BAUD 0xF400 // 2400bps @ 22.1184MHz
#define BAUD 0xFA00 // 4800bps @ 22.1184MHz
#define BAUD 0xFD00 // 9600bps @ 22.1184MHz
#define BAUD 0xFE80 // 19200bps @ 22.1184MHz
#define BAUD 0xFF40 // 38400bps @ 22.1184MHz
#define BAUD 0xFF80 // 57600bps @ 22.1184MHz
```

---

```

sfr AUXR = 0x8E;
sbit RXB = P3^0;                                //define UART TX/RX port
sbit TXB = P3^1;

typedef bit BOOL;
typedef unsigned char BYTE;
typedef unsigned int WORD;

BYTE TBUF,RBUF;
BYTE TDAT,RDAT;
BYTE TCNT,RCNT;
BYTE TBIT,RBIT;
BOOL TING,RING;
BOOL TEND,REND;

void UART_INIT();

BYTE t, r;
BYTE buf[16];

void main()
{
    TMOD = 0x00;                                //timer1 in 16-bit auto reload mode
    AUXR = 0x40;                                //timer1 working at 1T mode
    TL1 = BAUD;
    TH1 = BAUD>>8;                             //initial timer1 and set reload value
    TR1 = 1;                                    //tiemr1 start running
    ET1 = 1;                                    //enable timer1 interrupt
    PT1 = 1;                                    //improve timer1 interrupt priority
    EA = 1;                                     //open global interrupt switch

    UART_INIT();
    while (1)
    {
        //user's function
        if (REND)
        {
            REND = 0;
            buf[r++ & 0x0f] = RBUF;
        }
        if (TEND)
        {
            if (t != r)
            {
                TEND = 0;
                TBUF = buf[t++ & 0x0f];
                TING = 1;
            }
        }
    }
}

```

---

---

```

//-----
//Timer interrupt routine for UART

void tm1() interrupt 3 using 1
{
    if (RING)
    {
        if (--RCNT == 0)
        {
            RCNT = 3;           //reset send baudrate counter
            if (--RBIT == 0)
            {
                RBUF = RDAT;    //save the data to RBUF
                RING = 0;        //stop receive
                REND = 1;        //set receive completed flag
            }
            else
            {
                RDAT >>= 1;
                if (RXB) RDAT |= 0x80;    //shift RX data to RX buffer
            }
        }
    }
    else if (!RXB)
    {
        RING = 1;               //set start receive flag
        RCNT = 4;               //initial receive baudrate counter
        RBIT = 9;               //initial receive bit number (8 data bits + 1 stop bit)
    }

    if (--TCNT == 0)
    {
        TCNT = 3;               //reset send baudrate counter
        if (TING)               //judge whether sending
        {
            if (TBIT == 0)
            {
                TXB = 0;         //send start bit
                TDAT = TBUF;      //load data from TBUF to TDAT
                TBIT = 9;         //initial send bit number (8 data bits + 1 stop bit)
            }
        }
    }
}

```

---

---

```

        else
        {
            TDAT >>= 1;        //shift data to CY
            if (--TBIT == 0)
            {
                TXB = 1;
                TING = 0;        //stop send
                TEND = 1;        //set send completed flag
            }
            else
            {
                TXB = CY;        //write CY to TX port
            }
        }
    }
}

//-----
//initial UART module variable

void UART_INIT()
{
    TING = 0;
    RING = 0;
    TEND = 1;
    REND = 0;
    TCNT = 0;
    RCNT = 0;
}

```

---

## Assembly language code listing:

```
/*-----*/
/* --- STC MCU International Limited -----*/
/* --- STC 15 Series I/O simulate serial port -----*/
/* If you want to use the program or the program referenced in the */
/* article, please specify in which data and procedures from STC */
/*-----*/

;-----
;define baudrate const
;BAUD = 65536 - FOSC/3/BAUDRATE/M (1T:M=1; 12T:M=12)
;NOTE: (FOSC/3/BAUDRATE) must be greater then 75, (RECOMMEND GREATER THEN 100)

;BAUD EQU 0F400H ; 1200bps @ 11.0592MHz
;BAUD EQU 0FA00H ; 2400bps @ 11.0592MHz
;BAUD EQU 0FD00H ; 4800bps @ 11.0592MHz
;BAUD EQU 0FE80H ; 9600bps @ 11.0592MHz
;BAUD EQU 0FF40H ; 19200bps @ 11.0592MHz
;BAUD EQU 0FFA0H ; 38400bps @ 11.0592MHz
;BAUD EQU 0FFC0H ; 57600bps @ 11.0592MHz

;BAUD EQU 0EC00H ; 1200bps @ 18.432MHz
;BAUD EQU 0F600H ; 2400bps @ 18.432MHz
;BAUD EQU 0FB00H ; 4800bps @ 18.432MHz
;BAUD EQU 0FD80H ; 9600bps @ 18.432MHz
;BAUD EQU 0FEC0H ; 19200bps @ 18.432MHz
;BAUD EQU 0FF60H ; 38400bps @ 18.432MHz
BAUD EQU 0FF95H ; 57600bps @ 18.432MHz

;BAUD EQU 0E800H ; 1200bps @ 22.1184MHz
;BAUD EQU 0F400H ; 2400bps @ 22.1184MHz
;BAUD EQU 0FA00H ; 4800bps @ 22.1184MHz
;BAUD EQU 0FD00H ; 9600bps @ 22.1184MHz
;BAUD EQU 0FE80H ; 19200bps @ 22.1184MHz
;BAUD EQU 0FF40H ; 38400bps @ 22.1184MHz
;BAUD EQU 0FF80H ; 57600bps @ 22.1184MHz
```

---

```

;-----
;define UART TX/RX port

RXB   BIT    P3.0
TXB   BIT    P3.1

;-----
;define SFR

AUXR   DATA  8EH

;-----
;define UART module variable

TBUF   DATA  08H           ;(R0) ready send data buffer (USER WRITE ONLY)
RBUF   DATA  09H           ;(R1) received data buffer (UAER READ ONLY)
TDAT   DATA  0AH           ;(R2) sending data buffer (RESERVED FOR UART MODULE)
RDAT   DATA  0BH           ;(R3) receiving data buffer (RESERVED FOR UART MODULE)
TCNT   DATA  0CH           ;(R4) send baudrate counter (RESERVED FOR UART MODULE)
RCNT   DATA  0DH           ;(R5) receive baudrate counter (RESERVED FOR UART MODULE)
TBIT   DATA  0EH           ;(R6) send bit counter (RESERVED FOR UART MODULE)
RBIT   DATA  0FH           ;(R7) receive bit counter (RESERVED FOR UART MODULE)

TING   BIT    20H.0         ;sending flag(USER WRITE"1"TO TRIGGER SEND DATA,CLEAR BY MODULE)
RING   BIT    20H.1         ; receiving flag (RESERVED FOR UART MODULE)
TEND   BIT    20H.2         ; sent flag (SET BY MODULE AND SHOULD USER CLEAR)
REND   BIT    20H.3         ; received flag (SET BY MODULE AND SHOULD USER CLEAR)

RPTR   DATA  21H           ;circular queue read pointer
WPTR   DATA  22H           ;circular queue write pointer
BUFFER DATA  23H           ;circular queue buffer (16 bytes)

;-----
                ORG    0000H
                LJMP   RESET

;-----
;Timer1 interrupt routine for UART

                ORG    001BH

                PUSH   ACC                ;4 save ACC
                PUSH   PSW                ;4 save PSW
                MOV     PSW,    #08H      ;3 using register group 1
L_UARTSTART:
;-----
                JB     RING,    L_RING    ;4 judge whether receiving
                JB     RXB,     L_REND    ; check start signal

```

---

---

```

L_RSTART:
    SETB    RING                ; set start receive flag
    MOV     R5,    #4           ; initial receive baudrate counter
    MOV     R7,    #9           ; initial receive bit number (8 data bits + 1 stop bit)
    SJMP    L_REND              ; end this time slice

L_RING:
    DJNZ    R5,    L_REND       ;4 judge whether sending
    MOV     R5,    #3           ;2 reset send baudrate counter

L_RBIT:
    MOV     C,     RXB          ;3 read RX port data
    MOV     A,     R3           ;1 and shift it to RX buffer
    RRC     A                  ;1
    MOV     R3,    A            ;2
    DJNZ    R7,    L_REND       ;4 judge whether the data have receive completed

L_RSTOP:
    RLC     A                  ; shift out stop bit
    MOV     R1,    A            ; save the data to RBUF
    CLR     RING              ; stop receive
    SETB    REND              ; set receive completed flag

L_REND:
;-----
L_TING:
    DJNZ    R4,    L_TEND       ;4 check send baudrate counter
    MOV     R4,    #3           ;2 reset it
    JNB     TING,    L_TEND     ;4 judge whether sending
    MOV     A,     R6           ;1 detect the sent bits
    JNZ     L_TBIT              ;3 "0" means start bit not sent

L_TSTART:
    CLR     TXB                ; send start bit
    MOV     TDAT,   R0          ; load data from TBUF to TDAT
    MOV     R6,    #9           ; initial send bit number (8 data bits + 1 stop bit)
    JMP     L_TEND              ; end this time slice

L_TBIT:
    MOV     A,     R2           ;1 read data in TDAT
    SETB    C                  ;1 shift in stop bit
    RRC     A                  ;1 shift data to CY
    MOV     R2,    A            ;2 update TDAT
    MOV     TXB,   C            ;4 write CY to TX port
    DJNZ    R6,    L_TEND       ;4 judge whether the data have send completed

L_TSTOP:
    CLR     TING                ; stop send
    SETB    TEND                ; set send completed flag

L_TEND:
;-----
L_UARTEND:
    POP     PSW                ;3 restore PSW
    POP     ACC                ;3 restore ACC
    RETI                        ;4 (69)

```

---

---

```

;-----
;initial UART module variable

UART_INIT:
    CLR    TING
    CLR    RING
    SETB   TEND
    CLR    REND
    CLR    A
    MOV    TCNT,A
    MOV    RCNT,A
    RET

;-----
;main program entry

RESET:
    MOV    R0,    #7FH        ;clear RAM
    CLR    A
    MOV    @R0,   A
    DJNZ   R0,    $-1
    MOV    SP,    #7FH        ;initial SP

;-----
;system initial
    MOV    TMOD,  #00H        ;timer1 in 16-bit auto reload mode
    MOV    AUXR,  #40H        ;timer1 working at 1T mode
    MOV    TL1,   #LOW BAUD   ;initial timer1 and
    MOV    TH1,   #HIGH BAUD  ;set reload value
    SETB   TR1              ;tiemr1 start running
    SETB   ET1              ;enable timer1 interrupt
    SETB   PT1              ;improve timer1 interrupt priority
    SETB   EA              ;open global interrupt switch
    LCALL  UART_INIT

;-----
MAIN:
    JNB    REND,  CHECKREND   ;if (REND)
    CLR    REND              ;{
    MOV    A,     RPTR        ;    REND = 0;
    INC    RPTR              ;    BUFFER[RPTR++ & 0xf] = RBUF;
    ANL    A,     #0FH        ;}
    ADD    A,     #BUFFER      ;
    MOV    R0,    A           ;
    MOV    @R0,   RBUF        ;

```

---

---

CHECKREND:

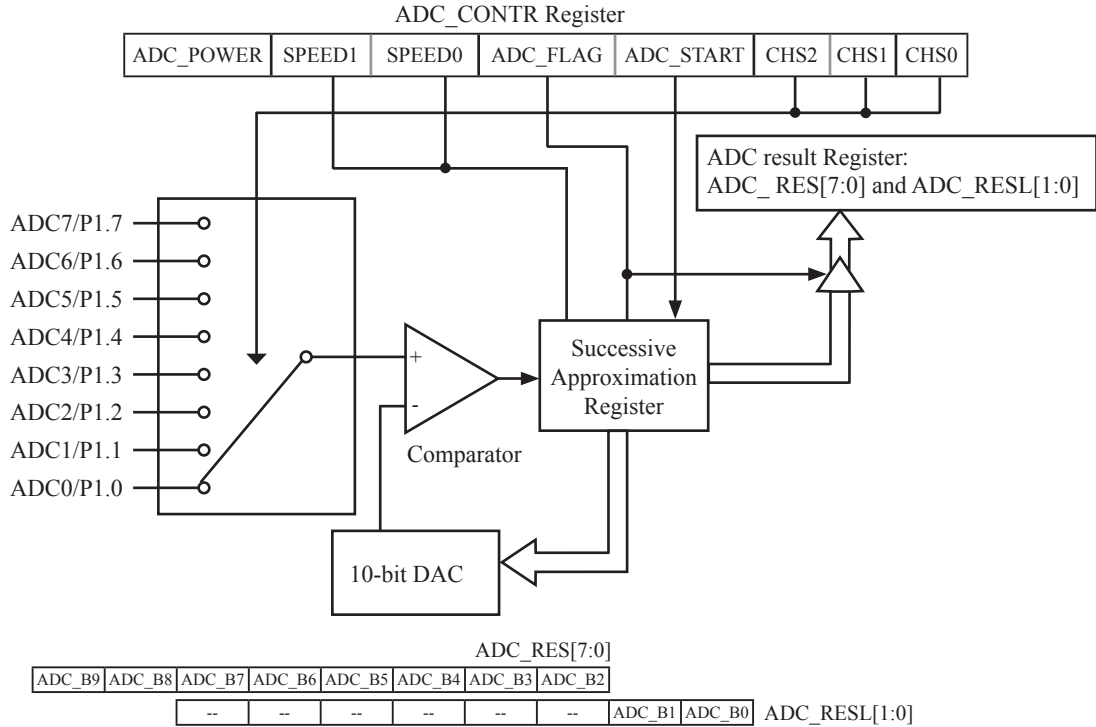
```
JNB    TEND,  MAIN          ;if (TEND)
MOV     A,    RPTR          ;{
XRL     A,    WPTR          ;    if (WPTR != REND)
JZ      MAIN          ;    {
CLR     TEND          ;                TEND = 0;
MOV     A,    WPTR          ;                TBUF = BUFFER[WPTR++ & 0xf];
INC     WPTR          ;                TING = 1;
ANL     A,    #0FH          ;                }
ADD     A,    #BUFFER          ;}
MOV     R0,    A          ;
MOV     TBUF,  @R0          ;
SETB    TING          ;
SJMP    MAIN
```

;-----

END

## Chapter 9 Analog to Digital Converter

### 9.1 A/D Converter Structure



The ADC on STC15F204EA series 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 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.

---

If users need 10-bit A/D Conversion result, They may be get the result from the following formula:

$$\text{10-bit A/D Conversion Result:}(\text{ADC\_RES}[7:0], \text{ADC\_RESL}[1:0]) = 1024 \times \frac{V_{in}}{V_{cc}}$$

If users only need 8-bit A/D Conversion result, They may be get the result from the following formula:

$$\text{8-bit A/D Conversion Result:}(\text{ADC\_RES}[7:0]) = 256 \times \frac{V_{in}}{V_{cc}}$$

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

## 9.2 Register for ADC

| SFR Name | SFR Address | bit  | B7     | B6     | B5     | B4     | B3     | B2     | B1     | B0     |
|----------|-------------|------|--------|--------|--------|--------|--------|--------|--------|--------|
| P1ASF    | 9DH         | name | 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

| SFR Name  | SFR Address | bit  | B7        | B6     | B5     | B4       | B3        | B2   | B1   | B0   |
|-----------|-------------|------|-----------|--------|--------|----------|-----------|------|------|------|
| ADC_CONTR | BCH         | 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\_START : 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 )**

| SFR name | Address | bit  | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 |
|----------|---------|------|----|----|----|----|----|----|----|----|
| ADC_RES  | BDH     | name |    |    |    |    |    |    |    |    |

The ADC\_RES is the final result from the A/D conversion

**ADC\_RESL(Low Byte of ADC result register )**

| SFR name | Address | bit  | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 |
|----------|---------|------|----|----|----|----|----|----|----|----|
| ADC_RESL | BEH     | name |    |    |    |    |    |    |    |    |

**IE: Interrupt Enable Register (Address:A8H)**

|       |      |      |   |       |     |     |     |
|-------|------|------|---|-------|-----|-----|-----|
| (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 (Address:B8H)**

|       |      |      |   |       |     |     |     |
|-------|------|------|---|-------|-----|-----|-----|
| (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. |
|------|------|-----------------------------|

---

### 9.3 Program using interrupts to demonstrate ADC

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

#### Assembly language code listing:

```
;/*----- */
/* --- STC MCU International Limited ----- */
/* --- STC 15 Series MCU A/D Conversion Demo ----- */
/* If you want to use the program or the program referenced in the -- */
/* article, please specify in which data and procedures from STC -- */
/*----- */

;define baudrate const
;BAUD = 65536 - FOSC/3/BAUDRATE/M (1T:M=1; 12T:M=12)
;NOTE: (FOSC/3/BAUDRATE) must be greater then 75, (RECOMMEND GREATER THEN 100)

;BAUD EQU 0F400H ; 1200bps @ 11.0592MHz
;BAUD EQU 0FA00H ; 2400bps @ 11.0592MHz
;BAUD EQU 0FD00H ; 4800bps @ 11.0592MHz
;BAUD EQU 0FE80H ; 9600bps @ 11.0592MHz
;BAUD EQU 0FF40H ; 19200bps @ 11.0592MHz
;BAUD EQU 0FFA0H ; 38400bps @ 11.0592MHz
;BAUD EQU 0FFC0H ; 57600bps @ 11.0592MHz
;BAUD EQU 0EC00H ; 1200bps @ 18.432MHz
;BAUD EQU 0F600H ; 2400bps @ 18.432MHz
;BAUD EQU 0FB00H ; 4800bps @ 18.432MHz
;BAUD EQU 0FD80H ; 9600bps @ 18.432MHz
;BAUD EQU 0FEC0H ; 19200bps @ 18.432MHz
;BAUD EQU 0FF60H ; 38400bps @ 18.432MHz
;BAUD EQU 0FF95H ; 57600bps @ 18.432MHz
;BAUD EQU 0E800H ; 1200bps @ 22.1184MHz
;BAUD EQU 0F400H ; 2400bps @ 22.1184MHz
;BAUD EQU 0FA00H ; 4800bps @ 22.1184MHz
;BAUD EQU 0FD00H ; 9600bps @ 22.1184MHz
;BAUD EQU 0FE80H ; 19200bps @ 22.1184MHz
;BAUD EQU 0FF40H ; 38400bps @ 22.1184MHz
;BAUD EQU 0FF80H ; 57600bps @ 22.1184MHz

;define UART TX/RX port
RXB BIT P3.0
TXB BIT P3.1

;define SFR
AUXR DATA 8EH
```

---

;define UART module variable

|      |      |       |                                                                       |
|------|------|-------|-----------------------------------------------------------------------|
| TBUF | DATA | 08H   | ; (R0) ready send data buffer (USER WRITE ONLY)                       |
| RBUF | DATA | 09H   | ; (R1) received data buffer (UAER READ ONLY)                          |
| TDAT | DATA | 0AH   | ; (R2) sending data buffer (RESERVED FOR UART MODULE)                 |
| RDAT | DATA | 0BH   | ; (R3) receiving data buffer (RESERVED FOR UART MODULE)               |
| TCNT | DATA | 0CH   | ; (R4) send baudrate counter (RESERVED FOR UART MODULE)               |
| RCNT | DATA | 0DH   | ; (R5) receive baudrate counter (RESERVED FOR UART MODULE)            |
| TBIT | DATA | 0EH   | ; (R6) send bit counter (RESERVED FOR UART MODULE)                    |
| RBIT | DATA | 0FH   | ; (R7) receive bit counter (RESERVED FOR UART MODULE)                 |
|      |      |       |                                                                       |
| TING | BIT  | 20H.0 | ; sending flag (USER WRITE "1" TO TRIGGER SEND DATA, CLEAR BY MODULE) |
| RING | BIT  | 20H.1 | ; receiving flag (RESERVED FOR UART MODULE)                           |
| TEND | BIT  | 20H.2 | ; sent flag (SET BY MODULE AND SHOULD USER CLEAR)                     |
| REND | BIT  | 20H.3 | ; received flag (SET BY MODULE AND SHOULD USER CLEAR)                 |

;/\*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 | 21H | ;ADC channel NO. |
|-------|------|-----|------------------|

;-----

|      |         |
|------|---------|
| ORG  | 0000H   |
| LJMP | MAIN    |
|      |         |
| ORG  | 000BH   |
| LJMP | TM0_ISR |
|      |         |
| ORG  | 002BH   |
| LJMP | ADC_ISR |

;-----

---

```

    ORG    0100H

MAIN:
    MOV     SP,      #7FH
    MOV     ADCCH, #0
    LCALL   INIT_UART           ;Init UART, use to show ADC result
    LCALL   INIT_ADC           ;Init ADC sfr
    MOV     TMOD,    #00H       ;timer0 in 16-bit auto reload mode
    MOV     AUXR,    #80H       ;timer0 working at 1T mode
    MOV     TL0,     #LOW BAUD  ;initial timer0 and
    MOV     TH0,     #HIGH BAUD ;set reload value
    SETB    TR0             ;tiemr0 start running
    MOV     IE,      #0A0H      ;Enable ADC interrupt and Open master interrupt switch
    SETB    ET0             ;enable timer0 interrupt
    SETB    PT0             ;improve timer0 interrupt priority
    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

; /*-----
;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

; /*-----
;Initial UART
;-----*/
INIT_UART:
    CLR    TING
    CLR    RING
    SETB   TEND
    CLR    REND
    CLR    A
    MOV    TCNT,  A
    MOV    RCNT,  A
    RET

; /*-----
;Send one byte data to PC
;Input: ACC (UART data)
;Output:-
;-----*/
SEND_DATA:
    JNB    TEND,  $
    CLR    TEND
    MOV    TBUF,  A
    SETB   TING
    RET

```

---

---

```

;-----
;Timer0 interrupt routine for UART

TM0_ISR:
    PUSH    ACC                ;4 save ACC
    PUSH    PSW                ;4 save PSW
    MOV     PSW,    #08H       ;3 using register group 1
L_UARTSTART:
;-----
    JB      RING,    L_RING     ;4 judge whether receiving
    JB      RXB,    L_REND      ; check start signal
L_RSTART:
    SETB    RING              ; set start receive flag
    MOV     R5,    #4          ; initial receive baudrate counter
    MOV     R7,    #9          ; initial receive bit number (8 data bits + 1 stop bit)
    SJMP    L_REND             ; end this time slice
L_RING:
    DJNZ    R5,    L_REND      ;4 judge whether sending
    MOV     R5,    #3          ;2 reset send baudrate counter
L_RBIT:
    MOV     C,      RXB        ;3 read RX port data
    MOV     A,      R3         ;1 and shift it to RX buffer
    RRC     A                ;1
    MOV     R3,     A          ;2
    DJNZ    R7,     L_REND     ;4 judge whether the data have receive completed
L_RSTOP:
    RLC     A                ; shift out stop bit
    MOV     R1,     A          ; save the data to RBUF
    CLR     RING           ; stop receive
    SETB    REND           ; set receive completed flag
L_REND:
;-----
L_TING:
    DJNZ    R4,     L_TEND     ;4 check send baudrate counter
    MOV     R4,     #3         ;2 reset it
    JNB     TING,    L_TEND     ;4 judge whether sending
    MOV     A,      R6         ;1 detect the sent bits
    JNZ     L_TBIT           ;3 "0" means start bit not sent
L_TSTART:
    CLR     TXB              ; send start bit
    MOV     TDAT,   R0         ; load data from TBUF to TDAT
    MOV     R6,     #9         ; initial send bit number (8 data bits + 1 stop bit)
    JMP     L_TEND           ; end this time slice
L_TBIT:
    MOV     A,      R2         ;1 read data in TDAT
    SETB    C                ;1 shift in stop bit
    RRC     A                ;1 shift data to CY
    MOV     R2,     A          ;2 update TDAT
    MOV     TXB,    C         ;4 write CY to TX port
    DJNZ    R6,     L_TEND     ;4 judge whether the data have send completed

```

---

---

```

L_TSTOP:
    CLR    TING                ; stop send
    SETB   TEND                ; set send completed flag
L_TEND:
;-----
L_UARTEND:
    POP    PSW                ;3 restore PSW
    POP    ACC                ;3 restore ACC
    RETI                     ;4 (69)
;-----

    END

```

### C language code listing:

```

/*-----*/
/* --- STC MCU International Limited -----*/
/* --- STC 15 Series MCU A/D Conversion Demo -----*/
/* 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 bit BOOL;
typedef unsigned char BYTE;
typedef unsigned int WORD;

//define baudrate const
//BAUD = 256 - FOSC/3/BAUDRATE/M (1T:M=1; 12T:M=12)
//NOTE: (FOSC/3/BAUDRATE) must be greater then 98, (RECOMMEND GREATER THEN 110)

#define BAUD 0xF400 // 1200bps @ 11.0592MHz
#define BAUD 0xFA00 // 2400bps @ 11.0592MHz
#define BAUD 0xFD00 // 4800bps @ 11.0592MHz
#define BAUD 0xFE80 // 9600bps @ 11.0592MHz
#define BAUD 0xFF40 // 19200bps @ 11.0592MHz
#define BAUD 0xFFA0 // 38400bps @ 11.0592MHz

```

---

```

#define BAUD 0xEC00 // 1200bps @ 18.432MHz
#define BAUD 0xF600 // 2400bps @ 18.432MHz
#define BAUD 0xFB00 // 4800bps @ 18.432MHz
#define BAUD 0xFD80 // 9600bps @ 18.432MHz
#define BAUD 0xFEC0 // 19200bps @ 18.432MHz
#define BAUD 0xFF60 // 38400bps @ 18.432MHz

#define BAUD 0xE800 // 1200bps @ 22.1184MHz
#define BAUD 0xF400 // 2400bps @ 22.1184MHz
#define BAUD 0xFA00 // 4800bps @ 22.1184MHz
#define BAUD 0xFD00 // 9600bps @ 22.1184MHz
#define BAUD 0xFE80 // 19200bps @ 22.1184MHz
#define BAUD 0xFF40 // 38400bps @ 22.1184MHz
#define BAUD 0xFF80 // 57600bps @ 22.1184MHz

sfr AUXR = 0x8E;
sbit RXB = P3^0; //define UART TX/RX port
sbit TXB = P3^1;

/*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 SendData(BYTE dat);
void Delay(WORD n);
void InitADC();

BYTE TBUF,RBUF;
BYTE TDATA,RDATA;
BYTE TCNT,RCNT;
BYTE TBIT,RBIT;
BOOL TING,RING;
BOOL TEND,REND;
BYTE ch = 0; //ADC channel NO.

```

---

---

```

void main()
{
    InitUart();           //Init UART, use to show ADC result
    InitADC();            //Init ADC sfr
    TMOD = 0x00;          //timer0 in 16-bit auto reload mode
    AUXR = 0x80;          //timer0 working at 1T mode
    TL0 = BAUD;
    TH0 = BAUD>>8;        //initial timer0 and set reload value
    TR0 = 1;              //tiemr0 start running
    IE = 0xa0;            //Enable ADC interrupt and Open master interrupt switch
    ET0 = 1;              //enable timer0 interrupt
    PT0 = 1;              //improve timer0 interrupt priority
                          //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
}

```

---

---

```

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

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

//-----
//Timer interrupt routine for UART

void tm0() interrupt 1 using 1
{
    if (RING)
    {
        if (--RCNT == 0)
        {
            RCNT = 3;           //reset send baudrate counter
            if (--RBIT == 0)
            {
                RBUF = RDAT;     //save the data to RBUF
                RING = 0;        //stop receive
                REND = 1;        //set receive completed flag
            }
            else
            {
                RDAT >>= 1;
                if (RXB) RDAT |= 0x80;    //shift RX data to RX buffer
            }
        }
    }
    else if (!RXB)
    {
        RING = 1;               //set start receive flag
        RCNT = 4;               //initial receive baudrate counter
        RBIT = 9;               //initial receive bit number (8 data bits + 1 stop bit)
    }
}

```

---

---

```

    if (--TCNT == 0)
    {
        TCNT = 3;           //reset send baudrate counter
        if (TING)           //judge whether sending
        {
            if (TBIT == 0)
            {
                TXB = 0;     //send start bit
                TDAT = TBUF;  //load data from TBUF to TDAT
                TBIT = 9;     //initial send bit number (8 data bits + 1 stop bit)
            }
            else
            {
                TDAT >>= 1;   //shift data to CY
                if (--TBIT == 0)
                {
                    TXB = 1;
                    TING = 0;   //stop send
                    TEND = 1;   //set send completed flag
                }
                else
                {
                    TXB = CY;   //write CY to TX port
                }
            }
        }
    }
}

//-----
//initial UART module variable
void InitUart()
{
    TING = 0;
    RING = 0;
    TEND = 1;
    REND = 0;
    TCNT = 0;
    RCNT = 0;
}

//-----
//initial UART module variable
void SendData(BYTE dat)
{
    while (!TEND);
    TEND = 0;
    TBUF = dat;
    TING = 1;
}

```

---

---

## 9.4 Program using polling to demonstrate ADC

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

### Assembly language code listing:

```
;/*-----*/
;/* --- STC MCU International Limited -----*/
;/* --- STC 15 Series MCU A/D Conversion Demo -----*/
;/* If you want to use the program or the program referenced in the ---*/
;/* article, please specify in which data and procedures from STC --*/
;/*-----*/

;define baudrate const
;BAUD = 65536 - FOSC/3/BAUDRATE/M (1T:M=1; 12T:M=12)
;NOTE: (FOSC/3/BAUDRATE) must be greater then 75, (RECOMMEND GREATER THEN 100)

;BAUD EQU 0F400H ; 1200bps @ 11.0592MHz
;BAUD EQU 0FA00H ; 2400bps @ 11.0592MHz
;BAUD EQU 0FD00H ; 4800bps @ 11.0592MHz
;BAUD EQU 0FE80H ; 9600bps @ 11.0592MHz
;BAUD EQU 0FF40H ; 19200bps @ 11.0592MHz
;BAUD EQU 0FFA0H ; 38400bps @ 11.0592MHz
;BAUD EQU 0FFC0H ; 57600bps @ 11.0592MHz
;BAUD EQU 0EC00H ; 1200bps @ 18.432MHz
;BAUD EQU 0F600H ; 2400bps @ 18.432MHz
;BAUD EQU 0FB00H ; 4800bps @ 18.432MHz
;BAUD EQU 0FD80H ; 9600bps @ 18.432MHz
;BAUD EQU 0FEC0H ; 19200bps @ 18.432MHz
;BAUD EQU 0FF60H ; 38400bps @ 18.432MHz
;BAUD EQU 0FF95H ; 57600bps @ 18.432MHz
;BAUD EQU 0E800H ; 1200bps @ 22.1184MHz
;BAUD EQU 0F400H ; 2400bps @ 22.1184MHz
;BAUD EQU 0FA00H ; 4800bps @ 22.1184MHz
;BAUD EQU 0FD00H ; 9600bps @ 22.1184MHz
;BAUD EQU 0FE80H ; 19200bps @ 22.1184MHz
;BAUD EQU 0FF40H ; 38400bps @ 22.1184MHz
;BAUD EQU 0FF80H ; 57600bps @ 22.1184MHz

;define UART TX/RX port
RXB BIT P3.0
TXB BIT P3.1

;define SFR
AUXR DATA 8EH
```

---

```

;define UART module variable
TBUF  DATA  08H      ;(R0) ready send data buffer  (USER WRITE ONLY)
RBUF  DATA  09H      ;(R1) received data buffer  (UAER READ ONLY)
TDAT  DATA  0AH      ;(R2) sending data buffer  (RESERVED FOR UART MODULE)
RDAT  DATA  0BH      ;(R3) receiving data buffer  (RESERVED FOR UART MODULE)
TCNT  DATA  0CH      ;(R4) send baudrate counter  (RESERVED FOR UART MODULE)
RCNT  DATA  0DH      ;(R5) receive baudrate counter (RESERVED FOR UART MODULE)
TBIT  DATA  0EH      ;(R6) send bit counter    (RESERVED FOR UART MODULE)
RBIT  DATA  0FH      ;(R7) receive bit counter   (RESERVED FOR UART MODULE)

TING  BIT    20H.0     ;sending flag(USER WRITE"1"TO TRIGGER SEND DATA,CLEAR BY MODULE)
RING  BIT    20H.1     ; receiving flag (RESERVED FOR UART MODULE)
TEND  BIT    20H.2     ; sent flag    (SET BY MODULE AND SHOULD USER CLEAR)
REND  BIT    20H.3     ; received flag (SET BY MODULE AND SHOULD USER CLEAR)

;*/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
PIASF     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      000BH
                LJMP     TM0_ISR
;-----
MAIN:
    MOV     SP,      #7FH
    MOV     TMOD,    #00H      ;timer0 in 16-bit auto reload mode
    MOV     AUXR,    #80H      ;timer0 working at 1T mode
    MOV     TL0,     #LOW BAUD ;initial timer0 and
    MOV     TH0,     #HIGH BAUD ;set reload value
    SETB    TR0       ;tiemr0 start running
    SETB    ET0       ;enable timer0 interrupt
    SETB    PT0       ;improve timer0 interrupt priority
    SETB    EA        ;open global interrupt switch

    LCALL   INIT_UART      ;Init UART, use to show ADC result
    LCALL   INIT_ADC       ;Init ADC sfr

```

---

---

```

;-----
    MOV    A,    #55H
    LCALL  SEND_DATA    ;Show result
    MOV    A,    #66H
    LCALL  SEND_DATA    ;Show result
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
    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:
    CLR    TING
    CLR    RING
    SETB   TEND
    CLR    REND
    CLR    A
    MOV    TCNT,    A
    MOV    RCNT,    A
    RET

```

---

---

```

; /*-----
;Send one byte data to PC
;Input: ACC (UART data)
;Output:-
;-----*/
SEND_DATA:
    JNB     TEND,  $
    CLR     TEND
    MOV     TBUF,  A
    SETB    TING
    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

;-----
;Timer0 interrupt routine for UART

TM0_ISR:
    PUSH    ACC                ;4 save ACC
    PUSH    PSW                ;4 save PSW
    MOV     PSW,  #08H         ;3 using register group 1
L_UARTSTART:
;-----
    JB      RING,  L_RING      ;4 judge whether receiving
    JB      RXB,   L_REND      ;check start signal
L_RSTART:
    SETB    RING                ; set start receive flag
    MOV     R5,    #4           ; initial receive baudrate counter
    MOV     R7,    #9           ; initial receive bit number (8 data bits + 1 stop bit)
    SJMP    L_REND              ; end this time slice
L_RING:
    DJNZ    R5,    L_REND      ;4 judge whether sending
    MOV     R5,    #3           ;2 reset send baudrate counter

```

---

---

```

L_RBIF:
    MOV     C,      RXB           ;3 read RX port data
    MOV     A,      R3           ;1 and shift it to RX buffer
    RRC     A                ;1
    MOV     R3,     A            ;2
    DJNZ    R7,     L_REND       ;4 judge whether the data have receive completed
L_RSTOP:
    RLC     A                ; shift out stop bit
    MOV     R1,     A            ; save the data to RBUF
    CLR     RING           ; stop receive
    SETB    REND           ; set receive completed flag
L_REND:
;-----
L_TING:
    DJNZ    R4,     L_TEND       ;4 check send baudrate counter
    MOV     R4,     #3          ;2 reset it
    JNB     TING,    L_TEND       ;4 judge whether sending
    MOV     A,      R6           ;1 detect the sent bits
    JNZ     L_TBIT             ;3 "0" means start bit not sent
L_TSTART:
    CLR     TXB             ; send start bit
    MOV     TDAT,    R0          ; load data from TBUF to TDAT
    MOV     R6,     #9          ; initial send bit number (8 data bits + 1 stop bit)
    JMP     L_TEND           ; end this time slice
L_TBIT:
    MOV     A,      R2          ;1 read data in TDAT
    SETB    C                ;1 shift in stop bit
    RRC     A                ;1 shift data to CY
    MOV     R2,     A            ;2 update TDAT
    MOV     TXB,    C            ;4 write CY to TX port
    DJNZ    R6,     L_TEND       ;4 judge whether the data have send completed
L_TSTOP:
    CLR     TING             ; stop send
    SETB    TEND             ; set send completed flag
L_TEND:
;-----
L_UARTEND:
    POP     PSW              ;3 restore PSW
    POP     ACC              ;3 restore ACC
    RETI                     ;4 (69)

;-----

    END

```

---

---

## C language code listing:

```
/*-----*/
/* --- STC MCU International Limited -----*/
/* --- STC 15 Series MCU A/D Conversion Demo -----*/
/* 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 bit BOOL;
typedef unsigned char BYTE;
typedef unsigned int WORD;

//define baudrate const
//BAUD = 256 - FOSC/3/BAUDRATE/M (1T:M=1; 12T:M=12)
//NOTE: (FOSC/3/BAUDRATE) must be greater then 98, (RECOMMEND GREATER THEN 110)

#define BAUD 0xF400 // 1200bps @ 11.0592MHz
#define BAUD 0xFA00 // 2400bps @ 11.0592MHz
#define BAUD 0xFD00 // 4800bps @ 11.0592MHz
#define BAUD 0xFE80 // 9600bps @ 11.0592MHz
#define BAUD 0xFF40 // 19200bps @ 11.0592MHz
#define BAUD 0xFFA0 // 38400bps @ 11.0592MHz

#define BAUD 0xEC00 // 1200bps @ 18.432MHz
#define BAUD 0xF600 // 2400bps @ 18.432MHz
#define BAUD 0xFB00 // 4800bps @ 18.432MHz
#define BAUD 0xFD80 // 9600bps @ 18.432MHz
#define BAUD 0xFEC0 // 19200bps @ 18.432MHz
#define BAUD 0xFF60 // 38400bps @ 18.432MHz

#define BAUD 0xE800 // 1200bps @ 22.1184MHz
#define BAUD 0xF400 // 2400bps @ 22.1184MHz
#define BAUD 0xFA00 // 4800bps @ 22.1184MHz
#define BAUD 0xFD00 // 9600bps @ 22.1184MHz
#define BAUD 0xFE80 // 19200bps @ 22.1184MHz
#define BAUD 0xFF40 // 38400bps @ 22.1184MHz
#define BAUD 0xFF80 // 57600bps @ 22.1184MHz
```

---

```

sfr AUXR = 0x8E;
sbit RXB = P3^0;           //define UART TX/RX port
sbit TXB = P3^1;

/*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

BYTE TBUF,RBUF;
BYTE TDAT,RDAT;
BYTE TCNT,RCNT;
BYTE TBIT,RBIT;
BOOL TING,RING;
BOOL TEND,REND;

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

void main()
{
    TMOD = 0x00;           //timer0 in 16-bit auto reload mode
    AUXR = 0x80;           //timer0 working at 1T mode
    TL0 = BAUD;
    TH0 = BAUD>>8;         //initial timer0 and set reload value
    TR0 = 1;               //tiemr0 start running
    ET0 = 1;               //enable timer0 interrupt
    PT0 = 1;               //improve timer0 interrupt priority
    EA = 1;                //open global interrupt switch

    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 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
    }

```

---

---

```

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

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

//-----
//Timer interrupt routine for UART

void tm0() interrupt 1 using 1
{
    if (RING)
    {
        if (--RCNT == 0)
        {
            RCNT = 3;                //reset send baudrate counter
            if (--RBIT == 0)
            {
                RBUF = RDAT;        //save the data to RBUF
                RING = 0;            //stop receive
                REND = 1;            //set receive completed flag
            }
            else
            {
                RDAT >>= 1;
                if (RXB) RDAT |= 0x80; //shift RX data to RX buffer
            }
        }
    }
    else if (!RXB)
    {
        RING = 1;                    //set start receive flag
        RCNT = 4;                    //initial receive baudrate counter
        RBIT = 9;                    //initial receive bit number (8 data bits + 1 stop bit)
    }
}

```

---

---

```

    if (--TCNT == 0)
    {
        TCNT = 3;           //reset send baudrate counter
        if (TING)           //judge whether sending
        {
            if (TBIT == 0)
            {
                TXB = 0;     //send start bit
                TDAT = TBUF;  //load data from TBUF to TDAT
                TBIT = 9;     //initial send bit number (8 data bits + 1 stop bit)
            }
            else
            {
                TDAT >>= 1;   //shift data to CY
                if (--TBIT == 0)
                {
                    TXB = 1;
                    TING = 0;  //stop send
                    TEND = 1;  //set send completed flag
                }
                else
                {
                    TXB = CY;  //write CY to TX port
                }
            }
        }
    }
}

//-----
//initial UART module variable
void InitUart()
{
    TING = 0;
    RING = 0;
    TEND = 1;
    REND = 0;
    TCNT = 0;
    RCNT = 0;
}

//-----
//initial UART module variable
void SendData(BYTE dat)
{
    while (!TEND);
    TEND = 0;
    TBUF = dat;
    TING = 1;
}

```

---

---

## Chapter 10 IAP / EEPROM

The ISP in STC15F204EA 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 10 pages(max). 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 STC15F204EA 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.

### 10.1 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 |      |       |          |     |     |     |     | Value after Power-on or Reset |
|-----------|--------------------------------|---------|------------------------|------|-------|----------|-----|-----|-----|-----|-------------------------------|
|           |                                |         | MSB                    |      |       |          | LSB |     |     |     |                               |
| 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  | POF      | GF1 | GF0 | PD  | IDL | xx11 0000B                    |

---

**IAP\_DATA: ISP/IAP Flash Data Register**

LSB

| SFR name | Address | bit  | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 |
|----------|---------|------|----|----|----|----|----|----|----|----|
| IAP_DATA | C2H     | name |    |    |    |    |    |    |    |    |

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

| SFR name  | Address | bit  | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 |
|-----------|---------|------|----|----|----|----|----|----|----|----|
| IAP_ADDRH | C3H     | name |    |    |    |    |    |    |    |    |

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

| SFR name  | Address | bit  | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 |
|-----------|---------|------|----|----|----|----|----|----|----|----|
| IAP_ADDRL | C4H     | name |    |    |    |    |    |    |    |    |

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

| SFR name | Address | bit  | B7 | B6 | B5 | B4 | B3 | B2 | B1  | B0  |
|----------|---------|------|----|----|----|----|----|----|-----|-----|
| IAP_CMD  | C5H     | name | -  | -  | -  | -  | -  | -  | 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

| SFR name | Address | bit  | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 |
|----------|---------|------|----|----|----|----|----|----|----|----|
| IAP_TRIG | C6H     | name |    |    |    |    |    |    |    |    |

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**

| SFR name  | Address | bit  | B7    | B6   | B5    | B4       | B3 | B2  | B1  | B0  |
|-----------|---------|------|-------|------|-------|----------|----|-----|-----|-----|
| IAP_CONTR | C7H     | name | 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<br>(2 SYSclks) | Program<br>=55uS | Sector Erase<br>=21mS | Recommended System<br>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.

---

## 10.2 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,<br>;Data ptr point to non-EEPROM area |
| ;MOV | IAP_ADDRL, | #0FFH      | ;Move 00 into address low-byte unit,<br>;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                               |

---

## 10.3 EEPROM Demo Programs written in Assembly Language

```
;/*-----*/
;/* --- STC MCU International Limited -----*/
;/* --- STC 1T Series MCU ISP/IAP/EEPROM Demo -----*/
;/* 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 STC15F204EA series 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:
LCALL  IAP_READ              ;Check whether all sector data is FF
                                ;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:
MOV    A,    R2               ;Program 512 bytes data into data flash
LCALL  IAP_PROGRAM           ;Ready IAP data
                                ;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:
LCALL  IAP_READ              ;Verify 512 bytes data
                                ;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

```

---

The following program is almost as same as the above except simulate UART has been used in it

```
;/*-----*/
;/* --- STC MCU International Limited -----*/
;/* --- STC 15 Series MCU ISP/IAP/EEPROM Demo -----*/
;/* If you want to use the program or the program referenced in the --*/
;/* article, please specify in which data and procedures from STC --*/
;/*-----*/

;-----
;define baudrate const
;BAUD = 65536 - FOSC/3/BAUDRATE/M (1T:M=1; 12T:M=12)
;NOTE: (FOSC/3/BAUDRATE) must be greater then 75, (RECOMMEND GREATER THEN 100)

;BAUD EQU 0F400H ; 1200bps @ 11.0592MHz
;BAUD EQU 0FA00H ; 2400bps @ 11.0592MHz
;BAUD EQU 0FD00H ; 4800bps @ 11.0592MHz
;BAUD EQU 0FE80H ; 9600bps @ 11.0592MHz
;BAUD EQU 0FF40H ; 19200bps @ 11.0592MHz
;BAUD EQU 0FFA0H ; 38400bps @ 11.0592MHz
;BAUD EQU 0FFC0H ; 57600bps @ 11.0592MHz

;BAUD EQU 0EC00H ; 1200bps @ 18.432MHz
;BAUD EQU 0F600H ; 2400bps @ 18.432MHz
;BAUD EQU 0FB00H ; 4800bps @ 18.432MHz
;BAUD EQU 0FD80H ; 9600bps @ 18.432MHz
;BAUD EQU 0FEC0H ; 19200bps @ 18.432MHz
;BAUD EQU 0FF60H ; 38400bps @ 18.432MHz
BAUD EQU 0FF95H ; 57600bps @ 18.432MHz

;BAUD EQU 0E800H ; 1200bps @ 22.1184MHz
;BAUD EQU 0F400H ; 2400bps @ 22.1184MHz
;BAUD EQU 0FA00H ; 4800bps @ 22.1184MHz
;BAUD EQU 0FD00H ; 9600bps @ 22.1184MHz
;BAUD EQU 0FE80H ; 19200bps @ 22.1184MHz
;BAUD EQU 0FF40H ; 38400bps @ 22.1184MHz
;BAUD EQU 0FF80H ; 57600bps @ 22.1184MHz
```

---

```

;-----
;define UART TX/RX port

RXB    BIT    P3.0
TXB    BIT    P3.1

;-----
;define SFR

AUXR    DATA  8EH
;-----
;define UART module variable

TBUF    DATA  08H    ;(R0) ready send data buffer  (USER WRITE ONLY)
RBUF    DATA  09H    ;(R1) received data buffer  (UAER READ ONLY)
TDAT    DATA  0AH    ;(R2) sending data buffer  (RESERVED FOR UART MODULE)
RDAT    DATA  0BH    ;(R3) receiving data buffer  (RESERVED FOR UART MODULE)
TCNT    DATA  0CH    ;(R4) send baudrate counter  (RESERVED FOR UART MODULE)
RCNT    DATA  0DH    ;(R5) receive baudrate counter (RESERVED FOR UART MODULE)
TBIT    DATA  0EH    ;(R6) send bit counter  (RESERVED FOR UART MODULE)
RBIT    DATA  0FH    ;(R7) receive bit counter  (RESERVED FOR UART MODULE)

TING    BIT    20H.0    ;sending flag
                ;(USER WRITE"1"TO TRIGGER SEND DATA, CLEAR BY MODULE)
RING    BIT    20H.1    ;receiving flag (RESERVED FOR UART MODULE)
TEND    BIT    20H.2    ;sent flag  (SET BY MODULE AND SHOULD USER CLEAR)
REND    BIT    20H.3    ;received flag (SET BY MODULE AND SHOULD USER CLEAR)

/*Declare SFR 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

```

---

---

```

; //EEPROM Start address
IAP_ADDRESS EQU 0800H
;-----
        ORG     0000H
        LJMP    MAIN

;-----
;Timer0 interrupt routine for UART

        ORG     000BH

        PUSH    ACC                ;4 save ACC
        PUSH    PSW                ;4 save PSW
        MOV     PSW, #08H          ;3 using register group 1
L_UARTSTART:
;-----
        JB      RING, L_RING        ;4 judge whether receiving
        JB      RXB, L_REND         ; check start signal
L_RSTART:
        SETB    RING                ; set start receive flag
        MOV     R5, #4               ; initial receive baudrate counter
        MOV     R7, #9               ; initial receive bit number (8 data bits + 1 stop bit)
        SJMP    L_REND              ; end this time slice
L_RING:
        DJNZ    R5, L_REND          ;4 judge whether sending
        MOV     R5, #3               ;2 reset send baudrate counter
L_RBIT:
        MOV     C, RXB               ;3 read RX port data
        MOV     A, R3                ;1 and shift it to RX buffer
        RRC     A                    ;1
        MOV     R3, A                ;2
        DJNZ    R7, L_REND          ;4 judge whether the data have receive completed
L_RSTOP:
        RLC     A                    ; shift out stop bit
        MOV     R1, A                ; save the data to RBUF
        CLR     RING                 ; stop receive
        SETB    REND                 ; set receive completed flag
L_REND:
;-----
L_TING:
        DJNZ    R4, L_TEND           ;4 check send baudrate counter
        MOV     R4, #3               ;2 reset it
        JNB     TING, L_TEND         ;4 judge whether sending
        MOV     A, R6                ;1 detect the sent bits
        JNZ     L_TBIT               ;3 "0" means start bit not sent

```

---

---

```

L_TSTART:
    CLR    TXB                ; send start bit
    MOV    TDAT, R0           ; load data from TBUF to TDAT
    MOV    R6, #9             ; initial send bit number (8 data bits + 1 stop bit)
    JMP    L_TEND             ; end this time slice

L_TBIT:
    MOV    A, R2              ;1 read data in TDAT
    SETB   C                  ;1 shift in stop bit
    RRC    A                  ;1 shift data to CY
    MOV    R2, A              ;2 update TDAT
    MOV    TXB, C             ;4 write CY to TX port
    DJNZ   R6, L_TEND         ;4 judge whether the data have send completed

L_TSTOP:
    CLR    TING               ; stop send
    SETB   TEND               ; set send completed flag

L_TEND:
;-----
L_UARTEND:
    POP    PSW                ;3 restore PSW
    POP    ACC                ;3 restore ACC
    RETI                      ;4 (69)

;-----
;initial UART module variable
UART_INIT:
    CLR    TING
    CLR    RING
    SETB   TEND
    CLR    REND
    CLR    A
    MOV    TCNT, A
    MOV    RCNT, A
    RET

;-----
;send UART data
UART_SEND:
    JNB    TEND, $
    CLR    TEND
    MOV    TBUF, A
    SETB   TING
    RET

```

---

---

```

;-----
ORG    0100H
MAIN:
MOV     SP,    #7FH
MOV     TMOD,  #00H           ;timer0 in 16-bit auto reload mode
MOV     AUXR,  #80H           ;timer0 working at 1T mode
MOV     TL0,   #LOW BAUD      ;initial timer0 and
MOV     TH0,   #HIGH BAUD     ;set reload value
SETB    TR0                   ;tiemr0 start running
SETB    ET0                   ;enable timer0 interrupt
SETB    PT0                   ;improve timer0 interrupt priority
SETB    EA                   ;open global interrupt switch
LCALL   UART_INIT

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:
LCALL   IAP_READ              ;Check whether all sector data is FF
LCALL   UART_SEND             ;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:
MOV     A,     R2              ;Program 512 bytes data into data flash
LCALL   IAP_PROGRAM            ;Ready IAP data
INC     DPTR                  ;Program flash
INC     R2                    ;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
    LCALL UART_SEND
    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

```

---

---

## 10.4 EEPROM Demo Program written in C Language

```
/*-----*/
/* --- STC MCU International Limited -----*/
/* --- STC 15 Series MCU ISP/IAP/EEPROM Demo -----*/
/* 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 STC15F204EA series 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();
}
```

---

The following C program is almost as same as the above, while simulate UART has been used in it.

```
/*-----*/
/* --- STC MCU International Limited -----*/
/* --- STC 15 Series MCU ISP/IAP/EEPROM Demo -----*/
/* 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 baudrate const
//BAUD = 256 - FOSC/3/BAUDRATE/M (1T:M=1; 12T:M=12)
//NOTE: (FOSC/3/BAUDRATE) must be greater then 98, (RECOMMEND GREATER THEN 110)

//define BAUD 0xF400 // 1200bps @ 11.0592MHz
//define BAUD 0xFA00 // 2400bps @ 11.0592MHz
//define BAUD 0xFD00 // 4800bps @ 11.0592MHz
//define BAUD 0xFE80 // 9600bps @ 11.0592MHz
//define BAUD 0xFF40 //19200bps @ 11.0592MHz
//define BAUD 0xFFA0 //38400bps @ 11.0592MHz

//define BAUD 0xEC00 // 1200bps @ 18.432MHz
//define BAUD 0xF600 // 2400bps @ 18.432MHz
//define BAUD 0xFB00 // 4800bps @ 18.432MHz
//define BAUD 0xFD80 // 9600bps @ 18.432MHz
//define BAUD 0xFEC0 //19200bps @ 18.432MHz
#define BAUD 0xFF60 //38400bps @ 18.432MHz

//define BAUD 0xE800 // 1200bps @ 22.1184MHz
//define BAUD 0xF400 // 2400bps @ 22.1184MHz
//define BAUD 0xFA00 // 4800bps @ 22.1184MHz
//define BAUD 0xFD00 // 9600bps @ 22.1184MHz
//define BAUD 0xFE80 //19200bps @ 22.1184MHz
//define BAUD 0xFF40 //38400bps @ 22.1184MHz
//define BAUD 0xFF80 //57600bps @ 22.1184MHz
```

---

```
sfr AUXR = 0x8E;
sbit RXB = P3^0;           //define UART TX/RX port
sbit TXB = P3^1;
```

```
typedef bit BOOL;
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
```

```
//EEPROM Start address
```

```
#define IAP_ADDRESS 0x800
```

```
BYTE TBUF,RBUF;
BYTE TDAT,RDAT;
BYTE TCNT,RCNT;
BYTE TBIT,RBIT;
BOOL TING,RING;
BOOL TEND,REND;
```

```
void UART_INIT();
void UART_SEND(BYTE dat);
```

---

```

void Delay(BYTE n);
void IapIdle();
BYTE IapReadByte(WORD addr);
void IapProgramByte(WORD addr, BYTE dat);
void IapEraseSector(WORD addr);

void main()
{
    WORD i;
    BYTE j;

    TMOD = 0x00;           //timer0 in 16-bit auto reload mode
    AUXR = 0x80;           //timer0 working at 1T mode
    TL0 = BAUD;
    TH0 = BAUD>>8;         //initial timer0 and set reload value
    TR0 = 1;               //tiemr0 start running
    ET0 = 1;               //enable timer0 interrupt
    PT0 = 1;               //improve timer0 interrupt priority
    EA = 1;                //open global interrupt switch
    UART_INIT();

    P1 = 0xfe;             //1111,1110 System Reset OK
    Delay(10);             //Delay
    UART_SEND(0x5a);
    UART_SEND(0xa5);
    IapEraseSector(IAP_ADDRESS); //Erase current sector
    for (i=0; i<512; i++)    //Check whether all sector data is FF
    {
        j = IapReadByte(IAP_ADDRESS+i);
        UART_SEND(j);
        // if (j != 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
        {
            j = IapReadByte(IAP_ADDRESS+i);
            UART_SEND(j);
            if (j != (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_ADDRH = addr;                        //Set ISP/IAP/EEPROM address low
    IAP_ADDRL = 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_ADDRH = addr;                        //Set ISP/IAP/EEPROM address low
    IAP_ADDRL = 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_ADDRH = addr;                        //Set ISP/IAP/EEPROM address low
    IAP_ADDRL = 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
        lapIdle();
    }
    //-----
    //Timer interrupt routine for UART

void tm0() interrupt 1 using 1
{
    if (RING)
    {
        if (--RCNT == 0)
        {
            RCNT = 3;           //reset send baudrate counter
            if (--RBIT == 0)
            {
                RBUF = RDAT;    //save the data to RBUF
                RING = 0;       //stop receive
                REND = 1;       //set receive completed flag
            }
            else
            {
                RDAT >>= 1;
                if (RXB) RDAT |= 0x80;    //shift RX data to RX buffer
            }
        }
    }
    else if (!RXB)
    {
        RING = 1;               //set start receive flag
        RCNT = 4;               //initial receive baudrate counter
        RBIT = 9;               //initial receive bit number (8 data bits + 1 stop bit)
    }

    if (--TCNT == 0)
    {
        TCNT = 3;               //reset send baudrate counter
        if (TING)               //judge whether sending
        {
            if (TBIT == 0)
            {
                TXB = 0;         //send start bit
                TDAT = TBUF;     //load data from TBUF to TDAT
                TBIT = 9;        //initial send bit number (8 data bits + 1 stop bit)
            }
        }
    }
}

```

---

---

```

        else
        {
            TDAT >>= 1;           //shift data to CY
            if (--TBIT == 0)
            {
                TXB = 1;
                TING = 0;          //stop send
                TEND = 1;          //set send completed flag
            }
            else
            {
                TXB = CY;          //write CY to TX port
            }
        }
    }
}

//-----
//initial UART module variable
void UART_INIT()
{
    TING = 0;
    RING = 0;
    TEND = 1;
    REND = 0;
    TCNT = 0;
    RCNT = 0;
}

//-----
//initial UART module variable
void UART_SEND(BYTE dat)
{
    while (!TEND);
    TEND = 0;
    TBUF = dat;
    TING = 1;
}

```

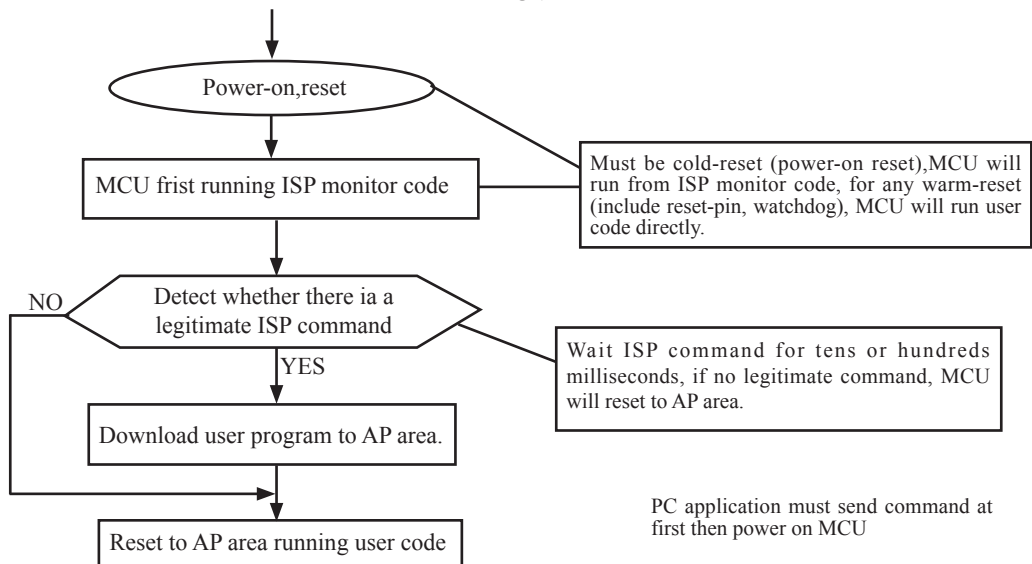
---

## Chapter 11 STC15Fxx series programming tools usage

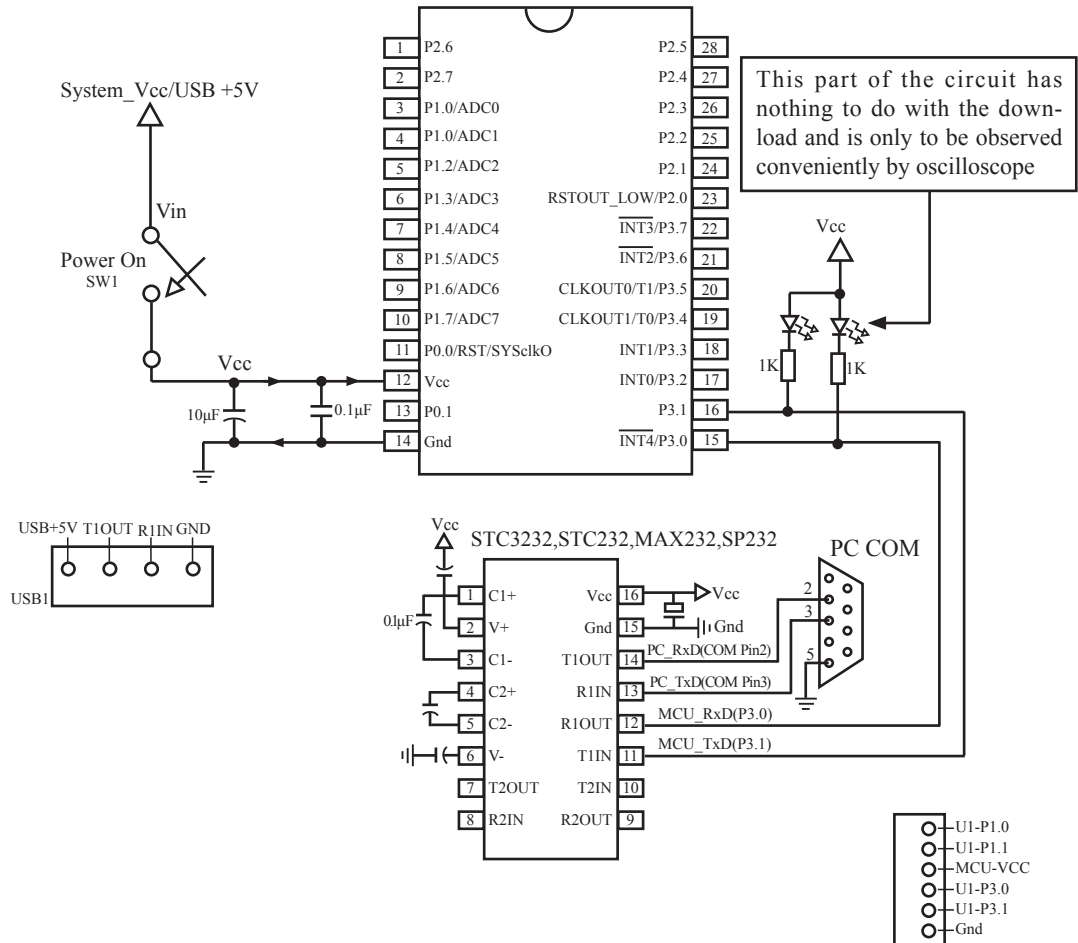
### 11.1 In-System-Programming (ISP) principle

If need download code into STC15F204EA series, P1.0 and P1.1 pin must be connected to GND

If you chose the "Next program code, P1.0/1.1 need=0/0" option, then the next time you need to re-download the program, first of all must be connected P1.0 and P1.1 to GND



## 11.2 STC15F204EA series application circuit for ISP



On-chip high-reliability Reset, No need external Reset circuit

Internal high-precision RC oscillator with temperature drifting  $\pm 1\%$ ( $-40^{\circ}\text{C} \sim +80^{\circ}\text{C}$ ), No need expensive external crystal oscillator

### 11.3 PC side application usage

STC-ISP.exe <http://www.STCMCU.com> Support Mobile:(86)1392

Step 1: Select MCU Type.

MCU Type:  AP Memory Range: 0000 - 03FF

Step2: Open code file and EEPROM file.

Start (HEX) Check Sum

☒ Clear Buffer before

☒ Clear Buffer before

Step 3: Select COM Port, Max Baud.

COM:  Max Baud:

If Connection failed, try Max Baud = Min Baud:

Step4: Active following options after Next-PowerUp/Cold Reset

MCU Clock: ☒ On-Chip R/C clock ☐ External Crystal/Clock

After Power-Up Reset, add extra Reset-Delay-Time ☒ YES ☐ NO

Oscillator Gain( <12MHz can select Low): ☒ High ☐ Low

Next Program Code, P1.0/P1.1: ☒ Not Related ☐ Need = 0/0

Next Program Code, clear EEPROM data: ☐ YES ☒ NO

Step 5: Click the Programming button then supply MCU power.

☐ Reload the target program file automatically before ISP-Programming each time, in order to debug easily.

☒ After the target program file is changed, automatically reload the file, then send ISP-Programming command.

OK Count:   Please pay attention to [www.MCU-Memory.com](http://www.MCU-Memory.com).

According to actual situation, the user selects the appropriate maximum baud rate

In practice, if P3.0/P3.1 already connected to a RS232/RS485 or other equipment, it is recommended that selection P1.0 / P1.1 = 0/0 can download options

Press this button when mass production

Enable the option in debugging stage

All new settings are valid in the next power-on.

---

Step1 : Select MCU type (E.g. STC15F204EA series)

Step2 : Load user program code (\*.bin or \*.hex)

Step3 : Select the serial port you are using

Step4 : Config the hardware option

Step5 : Press “ISP programming” or “Re-Programming” button to download user program

NOTE : Must press “ISP programming” or “Re-Programming” button first, then power on MCU, otherwise will cannot download.

#### About hardware connection

1. MCU RXD (P3.0) ---- RS232 ---- PC COM port TXD (Pin3)
2. MCU TXD (P3.1) ---- RS232 ---- PC COM port RXD (Pin2)
3. MCU GND-----PC COM port GND (Pin5)
4. RS232 : You can select STC232 / STC3232 / MAX232 / MAX3232 / ...

#### Using a demo board as a programmer

STC-ISP ver3.0A PCB can be welded into three kinds of circuits, respectively, support the STC's 16/20/28/32 pins MCU, the back plate of the download boards are affixed with labels, users need to pay special attention to. All the download board is welded 40-pin socket, the socket's 20-pin is ground line, all types of MCU should be put on the socket according to the way of alignment with the ground. The method of programming user code using download board as follow:

1. According to the type of MCU choose supply voltage,
  - A. For 5V MCU, using jumper JP1 to connect MCU-VCC to +5V pin
  - B. For 3V MCU, using jumper JP1 to connect MCU-VCC to +3.3V pin
2. Download cable (Provide by STC)
  - A. Connect DB9 serial connector to the computer's RS-232 serial interface
  - B. Plug the USB interface at the same side into your computer's USB port for power supply
  - C. Connect the USB interface at the other side into STC download board
3. Other interfaces do not need to connect.
4. In a non-pressed state to SW1, and MCU-VCC power LED off.
5. For SW3
  - P1.0/P1.1 = 1/1 when SW3 is non-pressed
  - P1.0/P1.1 = 0/0 when SW3 is pressedIf you have select the “Next program code, P1.0/P1.1 Need = 0/0” option, then SW3 must be in a pressed state
6. Put target MCU into the U1 socket, and locking socket
7. Press the “Download” button in the PC side application
8. Press SW1 switch in the download board
9. Close the demo board power supply and remove the MCU after download successfully.

---

## 11.4 Compiler / Assembler Programmer and Emulator

### About Compiler/Assembler

Any traditional compiler / assembler and the popular Keil are suitable for STC MCU. For selection MCU body, the traditional compiler / assembler, you can choose Intel's 8052 / 87C52 / 87C52 / 87C58 or Philips's P87C52 / P87C54/P87C58 in the traditional environment, in Keil environment, you can choose the types in front of the proposed or download the STC chips database file (STC.CDB) from the STC official website.

### About Programmer

You can use the STC specific ISP programmer. (Can be purchased from the STC or apply for free sample). Programmer can be used as demo board

### About Emulator

We do not provide specific emulator now. If you have a traditional 8051 emulator, you can use it to simulate STC MCU's some 8052 basic functions.

## 11.5 Self-Defined ISP download Demo

```
/*-----*/
/* --- STC MCU International Limited -----*/
/* --- STC 1T Series MCU using software to custom download code Demo-----*/
/* 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 <instrins.h>

sfr IAP_CONTR = 0xc7;
sbit MCU_Start_Led = P1^7;

#define Self_Define_ISP_Download_Command 0x22
#define RELOAD_COUNT 0xfb //18.432MHz,12T,SMOD=0,9600bps
//#define RELOAD_COUNT 0xf6 //18.432MHz,12T,SMOD=0,4800bps
//#define RELOAD_COUNT 0xec //18.432MHz,12T,SMOD=0,2400bps
//#define RELOAD_COUNT 0xd8 //18.432MHz,12T,SMOD=0,1200bps

void serial_port_initial(void);
void send_UART(unsigned char);
void UART_Interrupt_Receive(void);
void soft_reset_to_ISP_Monitor(void);
void delay(void);
void display_MCU_Start_Led(void);
```

---

```

void main(void)
{
    unsigned char i = 0;

    serial_port_initial();           //Initial UART
    display_MCU_Start_Led();         //Turn on the work LED
    send_UART(0x34);                 //Send UART test data
    send_UART(0xa7);                 // Send UART test data
    while (1);
}

void send_UART(unsigned char i)
{
    ES = 0;                         //Disable serial interrupt
    TI = 0;                         //Clear TI flag
    SBUF = i;                       //send this data
    while (!TI);                   //wait for the data is sent
    TI = 0;                         //clear TI flag
    ES = 1;                         //enable serial interrupt
}

void UART_Interrupt)Receive(void) interrupt 4 using 1
{
    unsigned char k = 0;
    if (RI)
    {
        RI = 0;
        k = SBUF;
        if (k == Self_Define_ISP_Command)           //check the serial data
        {
            delay();                                //delay 1s
            delay();                                //delay 1s
            soft_reset_to_ISP_Monitor();
        }
    }
    if (TI)
    {
        TI = 0;
    }
}

void soft_reset_to_ISP_Monitor(void)
{
    IAP_CONTR = 0x60;                             //0110,0000 soft reset system to run ISP monitor
}

```

---

---

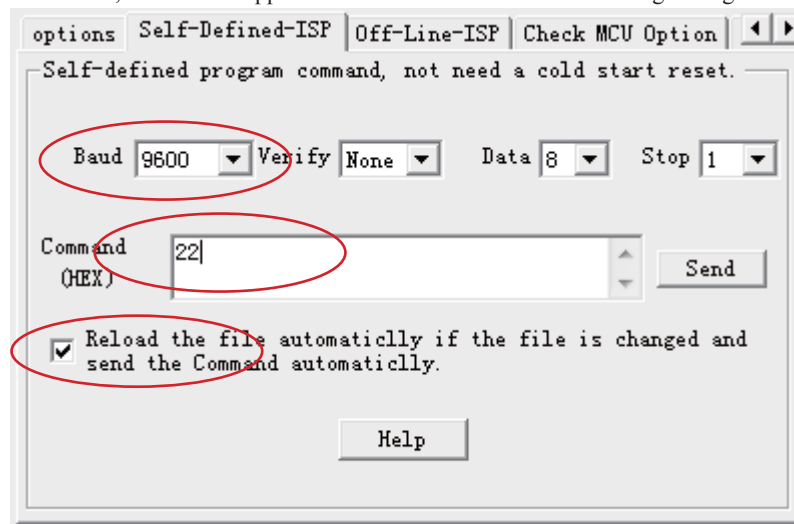
```

void delay(void)
{
    unsigned int j = 0;
    unsigned int g = 0;
    for (j=0; j<5; j++)
    {
        for (g=0; g<60000; g++)
        {
            _nop_();
            _nop_();
            _nop_();
            _nop_();
            _nop_();
        }
    }
}

void display_MCU_Start_Led(void)
{
    unsigned char i = 0;
    for (i=0; i<3; i++)
    {
        MCU_Start_Led = 0;      //Turn on work LED
        delay();
        MCU_Start_Led = 1;      //Turn off work LED
        delay();
        MCU_Start_Led = 0;      //Turn on work LED
    }
}

```

In addition, the PC-side application also need to make the following settings



---

## Appendix A: Assembly Language Programming

### INTRODUCTION

Assembly language is a computer language lying between the extremes of machine language and high-level language like Pascal or C use words and statements that are easily understood by humans, although still a long way from "natural" language. Machine language is the binary language of computers. A machine language program is a series of binary bytes representing instructions the computer can execute.

Assembly language replaces the binary codes of machine language with easy to remember "mnemonics" that facilitate programming. For example, an addition instruction in machine language might be represented by the code "10110011". It might be represented in assembly language by the mnemonic "ADD". Programming with mnemonics is obviously preferable to programming with binary codes.

Of course, this is not the whole story. Instructions operate on data, and the location of the data is specified by various "addressing modes" embedded in the binary code of the machine language instruction. So, there may be several variations of the ADD instruction, depending on what is added. The rules for specifying these variations are central to the theme of assembly language programming.

An assembly language program is not executable by a computer. Once written, the program must undergo translation to machine language. In the example above, the mnemonic "ADD" must be translated to the binary code "10110011". Depending on the complexity of the programming environment, this translation may involve one or more steps before an executable machine language program results. As a minimum, a program called an "assembler" is required to translate the instruction mnemonics to machine language binary codes. A further step may require a "linker" to combine portions of program from separate files and to set the address in memory at which the program may execute. We begin with a few definitions.

An assembly language program is a program written using labels, mnemonics, and so on, in which each statement corresponds to a machine instruction. Assembly language programs, often called source code or symbolic code, cannot be executed by a computer.

A machine language program is a program containing binary codes that represent instructions to a computer. Machine language programs, often called object code, are executable by a computer.

An assembler is a program that translates an assembly language program into a machine language program. The machine language program (object code) may be in "absolute" form or in "relocatable" form. In the latter case, "linking" is required to set the absolute address for execution.

A linker is a program that combines relocatable object programs (modules) and produces an absolute object program that is executable by a computer. A linker is sometimes called a "linker/locator" to reflect its separate functions of combining relocatable modules (linking) and setting the address for execution (locating).

A segment is a unit of code or data memory. A segment may be relocatable or absolute. A relocatable segment has a name, type, and other attributes that allow the linker to combine it with other partial segments, if required, and to correctly locate the segment. An absolute segment has no name and cannot be combined with other segments.

A module contains one or more segments or partial segments. A module has a name assigned by the user. The module definitions determine the scope of local symbols. An object file contains one or more modules. A module may be thought of as a "file" in many instances.

A program consists of a single absolute module, merging all absolute and relocatable segments from all input modules. A program contains only the binary codes for instructions (with address and data constants) that are understood by a computer.

---

## ASSEMBLER OPERATION

There are many assembler programs and other support programs available to facilitate the development of applications for the 8051 microcontroller. Intel's original MCS-51 family assembler, ASM51, is no longer available commercially. However, it set the standard to which the others are compared.

ASM51 is a powerful assembler with all the bells and whistles. It is available on Intel development systems and on the IBM PC family of microcomputers. Since these "host" computers contain a CPU chip other than the 8051, ASM51 is called a cross assembler. An 8051 source program may be written on the host computer (using any text editor) and may be assembled to an object file and listing file (using ASM51), but the program may not be executed. Since the host system's CPU chip is not an 8051, it does not understand the binary instruction in the object file. Execution on the host computer requires either hardware emulation or software simulation of the target CPU. A third possibility is to download the object program to an 8051-based target system for execution.

ASM51 is invoked from the system prompt by

ASM51 source\_file [assembler\_controls]

The source file is assembled and any assembler controls specified take effect. The assembler receives a source file as input (e.g., PROGRAM.SRC) and generates an object file (PROGRAM.OBJ) and listing file (PROGRAM.LST) as output. This is illustrated in Figure 1.

Since most assemblers scan the source program twice in performing the translation to machine language, they are described as two-pass assemblers. The assembler uses a location counter as the address of instructions and the values for labels. The action of each pass is described below.

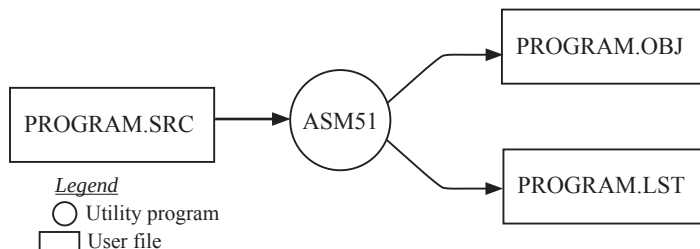


Figure 1 Assembling a source program

### Pass one

During the first pass, the source file is scanned line-by-line and a symbol table is built. The location counter defaults to 0 or is set by the ORG (set origin) directive. As the file is scanned, the location counter is incremented by the length of each instruction. Define data directives (DBs or DWs) increment the location counter by the number of bytes defined. Reserve memory directives (DSs) increment the location counter by the number of bytes reserved.

Each time a label is found at the beginning of a line, it is placed in the symbol table along with the current value of the location counter. Symbols that are defined using equate directives (EQUs) are placed in the symbol table along with the "equated" value. The symbol table is saved and then used during pass two.

### Pass two

During pass two, the object and listing files are created. Mnemonics are converted to opcodes and placed in the output files. Operands are evaluated and placed after the instruction opcodes. Where symbols appear in the operand field, their values are retrieved from the symbol table (created during pass one) and used in calculating the correct data or addresses for the instructions.

Since two passes are performed, the source program may use "forward references", that is, use a symbol before it is defined. This would occur, for example, in branching ahead in a program.

---

The object file, if it is absolute, contains only the binary bytes (00H-0FH) of the machine language program. A relocatable object file will also contain a symbol table and other information required for linking and locating. The listing file contains ASCII text codes (02H-7EH) for both the source program and the hexadecimal bytes in the machine language program.

A good demonstration of the distinction between an object file and a listing file is to display each on the host computer's CRT display (using, for example, the TYPE command on MS-DOS systems). The listing file clearly displays, with each line of output containing an address, opcode, and perhaps data, followed by the program statement from the source file. The listing file displays properly because it contains only ASCII text codes. Displaying the object file is a problem, however. The output will appear as "garbage", since the object file contains binary codes of an 8051 machine language program, rather than ASCII text codes.

## ASSEMBLY LANGUAGE PROGRAM FORMAT

Assembly language programs contain the following:

- Machine instructions
- Assembler directives
- Assembler controls
- Comments

Machine instructions are the familiar mnemonics of executable instructions (e.g., ANL). Assembler directives are instructions to the assembler program that define program structure, symbols, data, constants, and so on (e.g., ORG). Assembler controls set assembler modes and direct assembly flow (e.g., \$TITLE). Comments enhance the readability of programs by explaining the purpose and operation of instruction sequences.

Those lines containing machine instructions or assembler directives must be written following specific rules understood by the assembler. Each line is divided into "fields" separated by space or tab characters. The general format for each line is as follows:

```
[label:]    mnemonic    [operand]    [, operand]    [...]    [:comment]
```

Only the mnemonic field is mandatory. Many assemblers require the label field, if present, to begin on the left in column 1, and subsequent fields to be separated by space or tab characters. With ASM51, the label field needn't begin in column 1 and the mnemonic field needn't be on the same line as the label field. The operand field must, however, begin on the same line as the mnemonic field. The fields are described below.

### Label Field

A label represents the address of the instruction (or data) that follows. When branching to this instruction, this label is used in the operand field of the branch or jump instruction (e.g., SJMP SKIP).

Whereas the term "label" always represents an address, the term "symbol" is more general. Labels are one type of symbol and are identified by the requirement that they must terminate with a colon(:). Symbols are assigned values or attributes, using directives such as EQU, SEGMENT, BIT, DATA, etc. Symbols may be addresses, data constants, names of segments, or other constructs conceived by the programmer. Symbols do not terminate with a colon. In the example below, PAR is a symbol and START is a label (which is a type of symbol).

```
PAR      EQU      500                      ;"PAR" IS A SYMBOL WHICH
                                              ;REPRESENTS THE VALUE 500
START:    MOV      A,      #0FFH           ;"START" IS A LABEL WHICH
                                              ;REPRESENTS THE ADDRESS OF
                                              ;THE MOV INSTRUCTION
```

A symbol (or label) must begin with a letter, question mark, or underscore (\_); must be followed by letters, digit, "?", or "\_"; and can contain up to 31 characters. Symbols may use upper- or lowercase characters, but they are treated the same. Reserved words (mnemonics, operators, predefined symbols, and directives) may not be used.

---

## Mnemonic Field

Instruction mnemonics or assembler directives go into mnemonic field, which follows the label field. Examples of instruction mnemonics are ADD, MOV, DIV, or INC. Examples of assembler directives are ORG, EQU, or DB.

## Operand Field

The operand field follows the mnemonic field. This field contains the address or data used by the instruction. A label may be used to represent the address of the data, or a symbol may be used to represent a data constant. The possibilities for the operand field are largely dependent on the operation. Some operations have no operand (e.g., the RET instruction), while others allow for multiple operands separated by commas. Indeed, the possibilities for the operand field are numerous, and we shall elaborate on these at length. But first, the comment field.

## Comment Field

Remarks to clarify the program go into comment field at the end of each line. Comments must begin with a semicolon (;). Each line may be comment lines by beginning them with a semicolon. Subroutines and large sections of a program generally begin with a comment block—several lines of comments that explain the general properties of the section of software that follows.

## Special Assembler Symbols

Special assembler symbols are used for the register-specific addressing modes. These include A, R0 through R7, DPTR, PC, C and AB. In addition, a dollar sign (\$) can be used to refer to the current value of the location counter. Some examples follow.

```
SETB    C
INC      DPTR
JNB      TI, $
```

The last instruction above makes effective use of ASM51's location counter to avoid using a label. It could also be written as

```
HERE:    JNB      TI, HERE
```

## Indirect Address

For certain instructions, the operand field may specify a register that contains the address of the data. The commercial "at" sign (@) indicates address indirection and may only be used with R0, R1, the DPTR, or the PC, depending on the instruction. For example,

```
ADD      A, @R0
MOVC     A, @A+PC
```

The first instruction above retrieves a byte of data from internal RAM at the address specified in R0. The second instruction retrieves a byte of data from external code memory at the address formed by adding the contents of the accumulator to the program counter. Note that the value of the program counter, when the add takes place, is the address of the instruction following MOVC. For both instruction above, the value retrieved is placed into the accumulator.

## Immediate Data

Instructions using immediate addressing provide data in the operand field that become part of the instruction. Immediate data are preceded with a pound sign (#). For example,

---

```
CONSTANT    EQU    100
             MOV    A,    #0FEH
             ORL    40H,   #CONSTANT
```

All immediate data operations (except MOV DPTR,#data) require eight bits of data. The immediate data are evaluated as a 16-bit constant, and then the low-byte is used. All bits in the high-byte must be the same (00H or FFH) or the error message "value will not fit in a byte" is generated. For example, the following instructions are syntactically correct:

```
MOV    A,    #0FF00H
MOV    A,    #00FFH
```

But the following two instructions generate error messages:

```
MOV    A,    #0FE00H
MOV    A,    #01FFH
```

If signed decimal notation is used, constants from -256 to +255 may also be used. For example, the following two instructions are equivalent (and syntactically correct):

```
MOV    A,    #-256
MOV    A,    #0FF00H
```

Both instructions above put 00H into accumulator A.

### Data Address

Many instructions access memory locations using direct addressing and require an on-chip data memory address (00H to 7FH) or an SFR address (80H to 0FFH) in the operand field. Predefined symbols may be used for the SFR addresses. For example,

```
MOV    A,    45H
MOV    A,    SBUF           ;SAME AS MOV A, 99H
```

### Bit Address

One of the most powerful features of the 8051 is the ability to access individual bits without the need for masking operations on bytes. Instructions accessing bit-addressable locations must provide a bit address in internal data memory (00h to 7FH) or a bit address in the SFRs (80H to 0FFH).

There are three ways to specify a bit address in an instruction: (a) explicitly by giving the address, (b) using the dot operator between the byte address and the bit position, and (c) using a predefined assembler symbol. Some examples follow.

```
SETB   0E7H           ;EXPLICIT BIT ADDRESS
SETB   ACC.7           ;DOT OPERATOR (SAME AS ABOVE)
JNB     TI,    $        ;"TI" IS A PRE-DEFINED SYMBOL
JNB     99H,    $        ;(SAME AS ABOVE)
```

### Code Address

A code address is used in the operand field for jump instructions, including relative jumps (SJMP and conditional jumps), absolute jumps and calls (ACALL, AJMP), and long jumps and calls (LJMP, LCALL).

The code address is usually given in the form of a label.

ASM51 will determine the correct code address and insert into the instruction the correct 8-bit signed offset, 11-bit page address, or 16-bit long address, as appropriate.

---

## Generic Jumps and Calls

ASM51 allows programmers to use a generic JMP or CALL mnemonic. "JMP" can be used instead of SJMP, AJMP or LJMP; and "CALL" can be used instead of ACALL or LCALL. The assembler converts the generic mnemonic to a "real" instruction following a few simple rules. The generic mnemonic converts to the short form (for JMP only) if no forward references are used and the jump destination is within -128 locations, or to the absolute form if no forward references are used and the instruction following the JMP or CALL instruction is in the same 2K block as the destination instruction. If short or absolute forms cannot be used, the conversion is to the long form.

The conversion is not necessarily the best programming choice. For example, if branching ahead a few instructions, the generic JMP will always convert to LJMP even though an SJMP is probably better. Consider the following assembled instructions sequence using three generic jumps.

| LOC  | OBJ    | LINE | SOURCE                        |
|------|--------|------|-------------------------------|
| 1234 |        | 1    | ORG 1234H                     |
| 1234 | 04     | 2    | START: INC A                  |
| 1235 | 80FD   | 3    | JMP START ;ASSEMBLES AS SJMP  |
| 12FC |        | 4    | ORG START + 200               |
| 12FC | 4134   | 5    | JMP START ;ASSEMBLES AS AJMP  |
| 12FE | 021301 | 6    | JMP FINISH ;ASSEMBLES AS LJMP |
| 1301 | 04     | 7    | FINISH: INC A                 |
|      |        | 8    | END                           |

The first jump (line 3) assembles as SJMP because the destination is before the jump ( i.e., no forward reference) and the offset is less than -128. The ORG directive in line 4 creates a gap of 200 locations between the label START and the second jump, so the conversion on line 5 is to AJMP because the offset is too great for SJMP. Note also that the address following the second jump (12FEH) and the address of START (1234H) are within the same 2K page, which, for this instruction sequence, is bounded by 1000H and 17FFH. This criterion must be met for absolute addressing. The third jump assembles as LJMP because the destination (FINISH) is not yet defined when the jump is assembled (i.e., a forward reference is used). The reader can verify that the conversion is as stated by examining the object field for each jump instruction.

## ASSEMBLE-TIME EXPRESSION EVALUATION

Values and constants in the operand field may be expressed three ways: (a) explicitly (e.g., 0EFH), (b) with a predefined symbol (e.g., ACC), or (c) with an expression (e.g., 2 + 3). The use of expressions provides a powerful technique for making assembly language programs more readable and more flexible. When an expression is used, the assembler calculates a value and inserts it into the instruction.

All expression calculations are performed using 16-bit arithmetic; however, either 8 or 16 bits are inserted into the instruction as needed. For example, the following two instructions are the same:

|     |       |            |                            |
|-----|-------|------------|----------------------------|
| MOV | DPTR, | #04FFH + 3 |                            |
| MOV | DPTR, | #0502H     | ;ENTIRE 16-BIT RESULT USED |

If the same expression is used in a "MOV A,#data" instruction, however, the error message "value will not fit in a byte" is generated by ASM51. An overview of the rules for evaluating expressions follows.

---

## Number Bases

The base for numeric constants is indicated in the usual way for Intel microprocessors. Constants must be followed with "B" for binary, "O" or "Q" for octal, "D" or nothing for decimal, or "H" for hexadecimal. For example, the following instructions are the same:

```
MOV    A, #15H
MOV    A, #1111B
MOV    A, #0FH
MOV    A, #17Q
MOV    A, #15D
```

Note that a digit must be the first character for hexadecimal constants in order to differentiate them from labels (i.e., "0A5H" not "A5H").

## Character Strings

Strings using one or two characters may be used as operands in expressions. The ASCII codes are converted to the binary equivalent by the assembler. Character constants are enclosed in single quotes ('). Some examples follow.

```
CJNE    A, #'Q', AGAIN
SUBB    A, #'0'           ;CONVERT ASCII DIGIT TO BINARY DIGIT
MOV     DPTR, #'AB'
MOV     DPTR, #4142H      ;SAME AS ABOVE
```

## Arithmetic Operators

The arithmetic operators are

|     |                                   |
|-----|-----------------------------------|
| +   | addition                          |
| -   | subtraction                       |
| *   | multiplication                    |
| /   | division                          |
| MOD | modulo (remainder after division) |

For example, the following two instructions are same:

```
MOV     A, 10 + 10H
MOV     A, #1AH
```

The following two instructions are also the same:

```
MOV     A, #25 MOD 7
MOV     A, #4
```

Since the MOD operator could be confused with a symbol, it must be separated from its operands by at least one space or tab character, or the operands must be enclosed in parentheses. The same applies for the other operators composed of letters.

## Logical Operators

The logical operators are

|     |         |                  |
|-----|---------|------------------|
| OR  | logical | OR               |
| AND | logical | AND              |
| XOR | logical | Exclusive OR     |
| NOT | logical | NOT (complement) |

---

The operation is applied on the corresponding bits in each operand. The operator must be separated from the operands by space or tab characters. For example, the following two instructions are the same:

```
MOV    A, # '9' AND 0FH
MOV    A, #9
```

The NOT operator only takes one operand. The following three MOV instructions are the same:

```
THREE      EQU    3
MINUS_THREE EQU    -3
MOV        A,      # (NOT THREE) + 1
MOV        A,      #MINUS_THREE
MOV        A,      #11111101B
```

## Special Operators

The special operators are

```
SHR      shift right
SHL      shift left
HIGH     high-byte
LOW      low-byte
()       evaluate first
```

For example, the following two instructions are the same:

```
MOV    A, #8 SHL 1
MOV    A, #10H
```

The following two instructions are also the same:

```
MOV    A, #HIGH 1234H
MOV    A, #12H
```

## Relational Operators

When a relational operator is used between two operands, the result is always false (0000H) or true (FFFFH).

The operators are

```
EQ      =      equals
NE      <>     not equals
LT      <       less than
LE      <=      less than or equal to
GT      >       greater than
GE      >=      greater than or equal to
```

Note that for each operator, two forms are acceptable (e.g., "EQ" or "="). In the following examples, all relational tests are "true":

```
MOV    A, #5 = 5
MOV    A, #5 NE 4
MOV    A, # 'X' LT 'Z'
MOV    A, # 'X' >= 'X'
MOV    A, # $ > 0
MOV    A, #100 GE 50
```

---

So, the assembled instructions are equal to

MOV A, #0FFH

Even though expressions evaluate to 16-bit results (i.e., 0FFFFH), in the examples above only the low-order eight bits are used, since the instruction is a move byte operation. The result is not considered too big in this case, because as signed numbers the 16-bit value FFFFH and the 8-bit value FFH are the same (-1).

## Expression Examples

The following are examples of expressions and the values that result:

| Expression  | Result  |
|-------------|---------|
| 'B' - 'A'   | 0001H   |
| 8/3         | 0002H   |
| 155 MOD 2   | 0001H   |
| 4 * 4       | 0010H   |
| 8 AND 7     | 0000H   |
| NOT 1       | FFFEH   |
| 'A' SHL 8   | 4100H   |
| LOW 65535   | 00FFH   |
| (8 + 1) * 2 | 0012H   |
| 5 EQ 4      | 0000H   |
| 'A' LT 'B'  | FFFFH   |
| 3 <= 3      | FFFFHss |

A practical example that illustrates a common operation for timer initialization follows: Put -500 into Timer 1 registers TH1 and TL1. In using the HIGH and LOW operators, a good approach is

```
VALUE EQU -500
MOV TH1, #HIGH VALUE
MOV TL1, #LOW VALUE
```

The assembler converts -500 to the corresponding 16-bit value (FE0CH); then the HIGH and LOW operators extract the high (FEH) and low (0CH) bytes. as appropriate for each MOV instruction.

## Operator Precedence

The precedence of expression operators from highest to lowest is

```
( )
HIGH LOW
* / MOD SHL SHR
+ -
EQ NE LT LE GT GE = <> < <= > >=
NOT
AND
OR XOR
```

When operators of the same precedence are used, they are evaluated left to right.

Examples:

| Expression       | Value |
|------------------|-------|
| HIGH ('A' SHL 8) | 0041H |
| HIGH 'A' SHL 8   | 0000H |
| NOT 'A' - 1      | FFBFH |
| 'A' OR 'A' SHL 8 | 4141H |

---

## ASSEMBLER DIRECTIVES

Assembler directives are instructions to the assembler program. They are not assembly language instructions executable by the target microprocessor. However, they are placed in the mnemonic field of the program. With the exception of DB and DW, they have no direct effect on the contents of memory.

ASM51 provides several categories of directives:

- Assembler state control (ORG, END, USING)
- Symbol definition (SEGMENT, EQU, SET, DATA, IDATA, XDATA, BIT, CODE)
- Storage initialization/reservation (DS, DBIT, DB, DW)
- Program linkage (PUBLIC, EXTRN, NAME)
- Segment selection (RSEG, CSEG, DSEG, ISEG, ESEG, XSEG)

Each assembler directive is presented below, ordered by category.

### Assembler State Control

**ORG (Set Origin)**      The format for the ORG (set origin) directive is

ORG      expression

The ORG directive alters the location counter to set a new program origin for statements that follow. A label is not permitted. Two examples follow.

```
ORG      100H                              ;SET LOCATION COUNTER TO 100H
ORG      ($ + 1000H) AND 0F00H      ;SET TO NEXT 4K BOUNDARY
```

The ORG directive can be used in any segment type. If the current segment is absolute, the value will be an absolute address in the current segment. If a relocatable segment is active, the value of the ORG expression is treated as an offset from the base address of the current instance of the segment.

**End**              The format of the END directive is

END

END should be the last statement in the source file. No label is permitted and nothing beyond the END statement is processed by the assembler.

**Using**            The format of the END directive is

USING    expression

This directive informs ASM51 of the currently active register bank. Subsequent uses of the predefined symbolic register addresses AR0 to AR7 will convert to the appropriate direct address for the active register bank. Consider the following sequence:

```
USING    3
PUSH    AR7
USING    1
PUSH    AR7
```

The first push above assembles to PUSH 1FH (R7 in bank 3), whereas the second push assembles to PUSH 0FH (R7 in bank 1).

Note that USING does not actually switch register banks; it only informs ASM51 of the active bank. Executing 8051 instructions is the only way to switch register banks. This is illustrated by modifying the example above as follows:

---

```

MOV    PSW, #00011000B      ;SELECT REGISTER BANK 3
USING  3
PUSH   AR7                  ;ASSEMBLE TO PUSH 1FH
MOV    PSW, #00001000B      ;SELECT REGISTER BANK 1
USING  1
PUSH   AR7                  ;ASSEMBLE TO PUSH 0FH

```

## Symbol Definition

The symbol definition directives create symbols that represent segment, registers, numbers, and addresses. None of these directives may be preceded by a label. Symbols defined by these directives may not have been previously defined and may not be redefined by any means. The SET directive is the only exception. Symbol definition directives are described below.

**Segment** The format for the SEGMENT directive is shown below.

```

symbol      SEGMENT      segment_type

```

The symbol is the name of a relocatable segment. In the use of segments, ASM51 is more complex than conventional assemblers, which generally support only "code" and "data" segment types. However, ASM51 defines additional segment types to accommodate the diverse memory spaces in the 8051. The following are the defined 8051 segment types (memory spaces):

- CODE (the code segment)
- XDATA (the external data space)
- DATA (the internal data space accessible by direct addressing, 00H–07H)
- IDATA (the entire internal data space accessible by indirect addressing, 00H–07H)
- BIT (the bit space; overlapping byte locations 20H–2FH of the internal data space)

For example, the statement

```

EPROM      SEGMENT      CODE

```

declares the symbol EPROM to be a SEGMENT of type CODE. Note that this statement simply declares what EPROM is. To actually begin using this segment, the RSEG directive is used (see below).

**EQU (Equate)** The format for the EQU directive is

```

Symbol      EQU      expression

```

The EQU directive assigns a numeric value to a specified symbol name. The symbol must be a valid symbol name, and the expression must conform to the rules described earlier.

The following are examples of the EQU directive:

```

N27      EQU      27      ;SET N27 TO THE VALUE 27
HERE     EQU      $      ;SET "HERE" TO THE VALUE OF
                        ;THE LOCATION COUNTER
CR       EQU      0DH     ;SET CR (CARRIAGE RETURN) TO 0DH
MESSAGE: DB 'This is a message'
LENGTH  EQU      $ - MESSAGE ;"LENGTH" EQUALS LENGTH OF "MESSAGE"

```

**Other Symbol Definition Directives** The SET directive is similar to the EQU directive except the symbol may be redefined later, using another SET directive.

---

The DATA, IDATA, XDATA, BIT, and CODE directives assign addresses of the corresponding segment type to a symbol. These directives are not essential. A similar effect can be achieved using the EQU directive; if used, however, they evoke powerful type-checking by ASM51. Consider the following two directives and four instructions:

```
FLAG1      EQU      05H
FLAG2      BIT       05H
           SETB      FLAG1
           SETB      FLAG2
           MOV       FLAG1, #0
           MOV       FLAG2, #0
```

The use of FLAG2 in the last instruction in this sequence will generate a "data segment address expected" error message from ASM51. Since FLAG2 is defined as a bit address (using the BIT directive), it can be used in a set bit instruction, but it cannot be used in a move byte instruction. Hence, the error. Even though FLAG1 represents the same value (05H), it was defined using EQU and does not have an associated address space. This is not an advantage of EQU, but rather, a disadvantage. By properly defining address symbols for use in a specific memory space (using the directives BIT, DATA, XDATA, etc.), the programmer takes advantage of ASM51's powerful type-checking and avoids bugs from the misuse of symbols.

### Storage Initialization/Reservation

The storage initialization and reservation directives initialize and reserve space in either word, byte, or bit units. The space reserved starts at the location indicated by the current value of the location counter in the currently active segment. These directives may be preceded by a label. The storage initialization/reservation directives are described below.

**DS (Define Storage)**    The format for the DS (define storage) directive is

```
[label:]    DS        expression
```

The DS directive reserves space in byte units. It can be used in any segment type except BIT. The expression must be a valid assemble-time expression with no forward references and no relocatable or external references. When a DS statement is encountered in a program, the location counter of the current segment is incremented by the value of the expression. The sum of the location counter and the specified expression should not exceed the limitations of the current address space.

The following statement create a 40-byte buffer in the internal data segment:

```
           DSEG      AT      30H        ;PUT IN DATA SEGMENT (ABSOLUTE, INTERNAL)
LENGTH    EQU       40
BUFFER:    DS        LENGRH        ;40 BYTES RESERVED
```

The label BUFFER represents the address of the first location of reserved memory. For this example, the buffer begins at address 30H because "AT 30H" is specified with DSEG. The buffer could be cleared using the following instruction sequence:

```
           MOV       R7,        #LENGTH
           MOV       R0,        #BUFFER
LOOP:      MOV       @R0,       #0
           DJNZ      R7,        LOOP
           (continue)
```

---

To create a 1000-byte buffer in external RAM starting at 4000H, the following directives could be used:

```
XSTART      EQU    4000H
XLENGTH     EQU    1000
             XSEG    AT    XSTART
XBUFFER:     DS    XLENGTH
```

This buffer could be cleared with the following instruction sequence:

```
      MOV     DPTR, #XBUFFER
LOOP: CLR     A
      MOVX    @DPTR, A
      INC     DPTR
      MOV     A,    DPL
      CJNE    A,    #LOW (XBUFFER + XLENGTH + 1), LOOP
      MOV     A,    DPH
      CJNE    A,    #HIGH (XBUFFER + XLENGTH + 1), LOOP
      (continue)
```

This is an excellent example of a powerful use of ASM51's operators and assemble-time expressions. Since an instruction does not exist to compare the data pointer with an immediate value, the operation must be fabricated from available instructions. Two compares are required, one each for the high- and low-bytes of the DPTR. Furthermore, the compare-and-jump-if-not-equal instruction works only with the accumulator or a register, so the data pointer bytes must be moved into the accumulator before the CJNE instruction. The loop terminates only when the data pointer has reached XBUFFER + LENGTH + 1. (The "+1" is needed because the data pointer is incremented after the last MOVX instruction.)

**DBIT**            The format for the DBIT (define bit) directive is,

```
[label:]            DBIT    expression
```

The DBIT directive reserves space in bit units. It can be used only in a BIT segment. The expression must be a valid assemble-time expression with no forward references. When the DBIT statement is encountered in a program, the location counter of the current (BIT) segment is incremented by the value of the expression. Note that in a BIT segment, the basic unit of the location counter is bits rather than bytes. The following directives creat three flags in a absolute bit segment:

```
      BSEG                    ;BIT SEGMENT (ABSOLUTE)
KEFLAG:    DBIT    1        ;KEYBOARD STATUS
PRFLAG:    DBIT    1        ;PRINTER STATUS
DKFLAG:    DBIT    1        ;DISK STATUS
```

Since an address is not specified with BSEG in the example above, the address of the flags defined by DBIT could be determined (if one wishes to to so) by examining the symbol table in the .LST or .M51 files. If the definitions above were the first use of BSEG, then KBFLAG would be at bit address 00H (bit 0 of byte address 20H). If other bits were defined previously using BSEG, then the definitions above would follow the last bit defined.

**DB (Define Byte)**            The format for the DB (define byte) directive is,

```
[label:]            DB       expression [, expression] [...]
```

The DB directive initializes code memory with byte values. Since it is used to actually place data constants in code memory, a CODE segment must be active. The expression list is a series of one or more byte values (each of which may be an expression) separated by commas.

---

The DB directive permits character strings (enclosed in single quotes) longer than two characters as long as they are not part of an expression. Each character in the string is converted to the corresponding ASCII code. If a label is used, it is assigned the address of the first byte. For example, the following statements

```
                CSEG  AT      0100H
SQUARES:  DB      0, 1, 4, 9, 16, 25      ;SQUARES OF NUMBERS 0-5
MESSAGE:  DB      'Login:', 0             ;NULL-TERMINATED CHARACTER STRING
```

When assembled, result in the following hexadecimal memory assignments for external code memory:

| Address | Contents |
|---------|----------|
| 0100    | 00       |
| 0101    | 01       |
| 0102    | 04       |
| 0103    | 09       |
| 0104    | 10       |
| 0105    | 19       |
| 0106    | 4C       |
| 0107    | 6F       |
| 0108    | 67       |
| 0109    | 69       |
| 010A    | 6E       |
| 010B    | 3A       |
| 010C    | 00       |

**DW (Define Word)** The format for the DW (define word) directive is

```
[label:]      DW      expression      [, expression] [...]
```

The DW directive is the same as the DB directive except two memory locations (16 bits) are assigned for each data item. For example, the statements

```
CSEG  AT      200H
DW    $, 'A', 1234H, 2, 'BC'
```

result in the following hexadecimal memory assignments:

| Address | Contents |
|---------|----------|
| 0200    | 02       |
| 0201    | 00       |
| 0202    | 00       |
| 0203    | 41       |
| 0204    | 12       |
| 0205    | 34       |
| 0206    | 00       |
| 0207    | 02       |
| 0208    | 42       |
| 0209    | 43       |

## Program Linkage

Program linkage directives allow the separately assembled modules (files) to communicate by permitting intermodule references and the naming of modules. In the following discussion, a "module" can be considered a "file." (In fact, a module may encompass more than one file.)

---

**Public**        The format for the PUBLIC (public symbol) directive is

PUBLIC            symbol    [, symbol]   [...]

The PUBLIC directive allows the list of specified symbols to be known and used outside the currently assembled module. A symbol declared PUBLIC must be defined in the current module. Declaring it PUBLIC allows it to be referenced in another module. For example,

PUBLIC    INCHAR, OUTCHR, INLINE, OUTSTR

**Extrn**        The format for the EXTRN (external symbol) directive is

EXTRN            segment\_type (symbol [, symbol] [...], ...)

The EXTRN directive lists symbols to be referenced in the current module that are defined in other modules. The list of external symbols must have a segment type associated with each symbol in the list. (The segment types are CODE, XDATA, DATA, IDATA, BIT, and NUMBER. NUMBER is a type-less symbol defined by EQU.) The segment type indicates the way a symbol may be used. The information is important at link-time to ensure symbols are used properly in different modules.

The PUBLIC and EXTRN directives work together. Consider the two files, MAIN.SRC and MESSAGES.SRC. The subroutines HELLO and GOOD\_BYE are defined in the module MESSAGES but are made available to other modules using the PUBLIC directive. The subroutines are called in the module MAIN even though they are not defined there. The EXTRN directive declares that these symbols are defined in another module.

MAIN.SRC:

```
EXTRN            CODE (HELLO, GOOD_BYE)
...
CALL            HELLO
...
CALL            GOOD_BYE
...
END
```

MESSAGES.SRC:

```
                 PUBLIC            HELLO, GOOD_BYE
...
HELLO:            (begin subroutine)
...
                 RET
GOOD_BYE:        (begin subroutine)
...
                 RET
...
                 END
```

Neither MAIN.SRC nor MESSAGES.SRC is a complete program; they must be assembled separately and linked together to form an executable program. During linking, the external references are resolved with correct addresses inserted as the destination for the CALL instructions.

**Name**        The format for the NAME directive is

NAME    module\_name

---

All the usual rules for symbol names apply to module names. If a name is not provided, the module takes on the file name (without a drive or subdirectory specifier and without an extension). In the absence of any use of the NAME directive, a program will contain one module for each file. The concept of "modules," therefore, is somewhat cumbersome, at least for relatively small programming problems. Even programs of moderate size (encompassing, for example, several files complete with relocatable segments) needn't use the NAME directive and needn't pay any special attention to the concept of "modules." For this reason, it was mentioned in the definition that a module may be considered a "file," to simplify learning ASM51. However, for very large programs (several thousand lines of code, or more), it makes sense to partition the problem into modules, where, for example, each module may encompass several files containing routines having a common purpose.

## Segment Selection Directives

When the assembler encounters a segment selection directive, it diverts the following code or data into the selected segment until another segment is selected by a segment selection directive. The directive may select a previously defined relocatable segment or optionally create and select absolute segments.

**RSEG (Relocatable Segment)** The format for the RSEG (relocatable segment) directive is

RSEG                    segment\_name

Where "segment\_name" is the name of a relocatable segment previously defined with the SEGMENT directive. RSEG is a "segment selection" directive that diverts subsequent code or data into the named segment until another segment selection directive is encountered.

**Selecting Absolute Segments** RSEG selects a relocatable segment. An "absolute" segment, on the other hand, is selected using one of the directives:

CSEG    (AT address)  
DSEG    (AT address)  
ISEG    (AT address)  
BSEG    (AT address)  
XSEG    (AT address)

These directives select an absolute segment within the code, internal data, indirect internal data, bit, or external data address spaces, respectively. If an absolute address is provided (by indicating "AT address"), the assembler terminates the last absolute address segment, if any, of the specified segment type and creates a new absolute segment starting at that address. If an absolute address is not specified, the last absolute segment of the specified type is continued. If no absolute segment of this type was previously selected and the absolute address is omitted, a new segment is created starting at location 0. Forward references are not allowed and start addresses must be absolute.

Each segment has its own location counter, which is always set to 0 initially. The default segment is an absolute code segment; therefore, the initial state of the assembler is location 0000H in the absolute code segment. When another segment is chosen for the first time, the location counter of the former segment retains the last active value. When that former segment is reselected, the location counter picks up at the last active value. The ORG directive may be used to change the location counter within the currently selected segment.

## ASSEMBLER CONTROLS

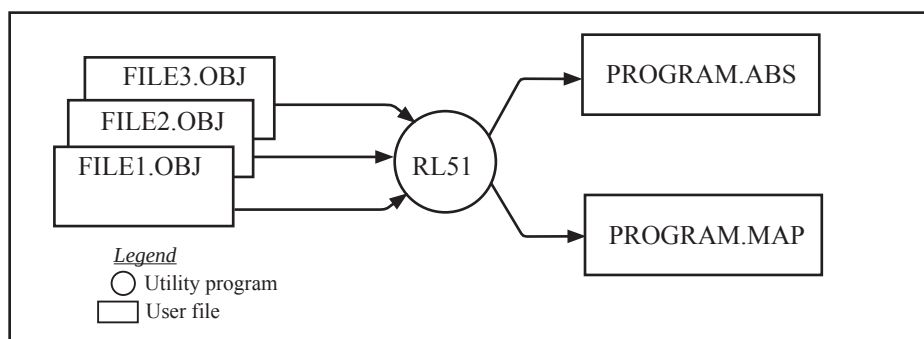
Assembler controls establish the format of the listing and object files by regulating the actions of ASM51. For the most part, assembler controls affect the look of the listing file, without having any affect on the program itself. They can be entered on the invocation line when a program is assembled, or they can be placed in the source file. Assembler controls appearing in the source file must be preceded with a dollar sign and must begin in column 1.

There are two categories of assembler controls: primary and general. Primary controls can be placed in the invocation line or at the beginning of the source program. Only other primary controls may precede a primary control. General controls may be placed anywhere in the source program.

## LINKER OPERATION

In developing large application programs, it is common to divide tasks into subprograms or modules containing sections of code (usually subroutines) that can be written separately from the overall program. The term "modular programming" refers to this programming strategy. Generally, modules are relocatable, meaning they are not intended for a specific address in the code or data space. A linking and locating program is needed to combine the modules into one absolute object module that can be executed.

Intel's RL51 is a typical linker/locator. It processes a series of relocatable object modules as input and creates an executable machine language program (PROGRAM, perhaps) and a listing file containing a memory map and symbol table (PROGRAM.M51). This is illustrated in following figure.



Linker operation

As relocatable modules are combined, all values for external symbols are resolved with values inserted into the output file. The linker is invoked from the system prompt by

```
RL51  input_list  [T0 output_file]  [location_controls]
```

The `input_list` is a list of relocatable object modules (files) separated by commas. The `output_list` is the name of the output absolute object module. If none is supplied, it defaults to the name of the first input file without any suffix. The `location_controls` set start addresses for the named segments.

For example, suppose three modules or files (MAIN.OBJ, MESSAGES.OBJ, and SUBROUTINES.OBJ) are to be combined into an executable program (EXAMPLE), and that these modules each contain two relocatable segments, one called EPROM of type CODE, and the other called ONCHIP of type DATA. Suppose further that the code segment is to be executable at address 4000H and the data segment is to reside starting at address 30H (in internal RAM). The following linker invocation could be used:

```
RS51  MAIN.OBJ, MESSAGES.OBJ, SUBROUTINES.OBJ TO EXAMPLE & CODE
      (EPROM (4000H) DATA (ONCHIP (30H))
```

Note that the ampersand character "&" is used as the line continuation character.

If the program begins at the label START, and this is the first instruction in the MAIN module, then execution begins at address 4000H. If the MAIN module was not linked first, or if the label START is not at the beginning of MAIN, then the program's entry point can be determined by examining the symbol table in the listing file EXAMPLE.M51 created by RL51. By default, EXAMPLE.M51 will contain only the link map. If a symbol table is desired, then each source program must have used the SDEBUG control. The following table shows the assembler controls supported by ASM51.

| Assembler controls supported by ASM51 |                     |                    |         |                                                                                                               |
|---------------------------------------|---------------------|--------------------|---------|---------------------------------------------------------------------------------------------------------------|
| NAME                                  | PRIMARY/<br>GENERAL | DEFAULT            | ABBREV. | MEANING                                                                                                       |
| DATE (date)                           | P                   | DATE( )            | DA      | Place string in header (9 char. max.)                                                                         |
| DEBUG                                 | P                   | NODEBUG            | DB      | Outputs debug symbol information to object file                                                               |
| EJECT                                 | G                   | not applicable     | EJ      | Continue listing on next page                                                                                 |
| ERRORPRINT<br>(file)                  | P                   | NOERRORPRINT       | EP      | Designates a file to receive error messages in addition to the listing file (defaults to console)             |
| NOERRORPRINT                          | P                   | NOERRORPRINT       | NOEP    | Designates that error messages will be printed in listing file only                                           |
| GEN                                   | G                   | GENONLY            | GO      | List only the fully expanded source as if all lines generated by a macro call were already in the source file |
| GENONLY                               | G                   | GENONLY            | NOGE    | List only the original source text in the listing file                                                        |
| INCLUDED(file)                        | G                   | not applicable     | IC      | Designates a file to be included as part of the program                                                       |
| LIST                                  | G                   | LIST               | LI      | Print subsequent lines of source code in listing file                                                         |
| NOLIST                                | G                   | LIST               | NOLI    | Do not print subsequent lines of source code in listing file                                                  |
| MACRO<br>(men_percent)                | P                   | MACRO(50)          | MR      | Evaluate and expand all macro calls. Allocate percentage of free memory for macro processing                  |
| NOMACRO                               | P                   | MACRO(50)          | NOMR    | Do not evaluate macro calls                                                                                   |
| MOD51                                 | P                   | MOD51              | MO      | Recognize the 8051-specific predefined special function registers                                             |
| NOMOD51                               | P                   | MOD51              | NOMO    | Do not recognize the 8051-specific predefined special function registers                                      |
| OBJECT(file)                          | P                   | OBJECT(source.OBJ) | OJ      | Designates file to receive object code                                                                        |
| NOOBJECT                              | P                   | OBJECT(source.OBJ) | NOOJ    | Designates that no object file will be created                                                                |
| PAGING                                | P                   | PAGING             | PI      | Designates that listing file be broken into pages and each will have a header                                 |
| NOPAGING                              | P                   | PAGING             | NOPI    | Designates that listing file will contain no page breaks                                                      |
| PAGELNGTH<br>(N)                      | P                   | PAGELNGT(60)       | PL      | Sets maximum number of lines in each page of listing file (range=10 to 65536)                                 |
| PAGE WIDTH (N)                        | P                   | PAGEWIDTH(120)     | PW      | Set maximum number of characters in each line of listing file (range = 72 to 132)                             |
| PRINT(file)                           | P                   | PRINT(source.LST)  | PR      | Designates file to receive source listing                                                                     |
| NOPRINT                               | P                   | PRINT(source.LST)  | NOPR    | Designates that no listing file will be created                                                               |
| SAVE                                  | G                   | not applicable     | SA      | Stores current control settings from SAVE stack                                                               |
| RESTORE                               | G                   | not applicable     | RS      | Restores control settings from SAVE stack                                                                     |
| REGISTERBANK<br>(rb,...)              | P                   | REGISTERBANK(0)    | RB      | Indicates one or more banks used in program module                                                            |
| NOREGISTER-<br>BANK                   | P                   | REGISTERBANK(0)    | NORB    | Indicates that no register banks are used                                                                     |
| SYMBOLS                               | P                   | SYMBOLS            | SB      | Creates a formatted table of all symbols used in program                                                      |
| NOSYMBOLS                             | P                   | SYMBOLS            | NOSB    | Designates that no symbol table is created                                                                    |
| TITLE(string)                         | G                   | TITLE( )           | TT      | Places a string in all subsequent page headers (max.60 characters)                                            |
| WORKFILES<br>(path)                   | P                   | same as source     | WF      | Designates alternate path for temporary workfiles                                                             |
| XREF                                  | P                   | NOXREF             | XR      | Creates a cross reference listing of all symbols used in program                                              |
| NOXREF                                | P                   | NOXREF             | NOXR    | Designates that no cross reference list is created                                                            |

---

## MACROS

The macro processing facility (MPL) of ASM51 is a "string replacement" facility. Macros allow frequently used sections of code be defined once using a simple mnemonic and used anywhere in the program by inserting the mnemonic. Programming using macros is a powerful extension of the techniques described thus far. Macros can be defined anywhere in a source program and subsequently used like any other instruction. The syntax for macro definition is

```
%*DEFINE      (call_pattern)      (macro_body)
```

Once defined, the call pattern is like a mnemonic; it may be used like any assembly language instruction by placing it in the mnemonic field of a program. Macros are made distinct from "real" instructions by preceding them with a percent sign, "%". When the source program is assembled, everything within the macro-body, on a character-by-character basis, is substituted for the call-pattern. The mystique of macros is largely unfounded. They provide a simple means for replacing cumbersome instruction patterns with primitive, easy-to-remember mnemonics. The substitution, we reiterate, is on a character-by-character basis—nothing more, nothing less.

For example, if the following macro definition appears at the beginning of a source file,

```
%*DEFINE      (PUSH_DPTR)
                (PUSH   DPH
                 PUSH   DPL
                 )
```

then the statement

```
%PUSH_DPTR
```

will appear in the .LST file as

```
PUSH   DPH
PUSH   DPL
```

The example above is a typical macro. Since the 8051 stack instructions operate only on direct addresses, pushing the data pointer requires two PUSH instructions. A similar macro can be created to POP the data pointer.

There are several distinct advantages in using macros:

- A source program using macros is more readable, since the macro mnemonic is generally more indicative of the intended operation than the equivalent assembler instructions.
- The source program is shorter and requires less typing.
- Using macros reduces bugs
- Using macros frees the programmer from dealing with low-level details.

The last two points above are related. Once a macro is written and debugged, it is used freely without the worry of bugs. In the PUSH\_DPTR example above, if PUSH and POP instructions are used rather than push and pop macros, the programmer may inadvertently reverse the order of the pushes or pops. (Was it the high-byte or low-byte that was pushed first?) This would create a bug. Using macros, however, the details are worked out once—when the macro is written—and the macro is used freely thereafter, without the worry of bugs.

Since the replacement is on a character-by-character basis, the macro definition should be carefully constructed with carriage returns, tabs, ect., to ensure proper alignment of the macro statements with the rest of the assembly language program. Some trial and error is required.

There are advanced features of ASM51's macro-processing facility that allow for parameter passing, local labels, repeat operations, assembly flow control, and so on. These are discussed below.

---

## Parameter Passing

A macro with parameters passed from the main program has the following modified format:

```
%*DEFINE      (macro_name (parameter_list)) (macro_body)
```

For example, if the following macro is defined,

```
%*DEFINE      (CMPA# (VALUE))  
              (CJNE  A, #%VALUE, $ + 3  
               )
```

then the macro call

```
%CMPA# (20H)
```

will expand to the following instruction in the .LST file:

```
CJNE  A, #20H, $ + 3
```

Although the 8051 does not have a "compare accumulator" instruction, one is easily created using the CJNE instruction with "\$+3" (the next instruction) as the destination for the conditional jump. The CMPA# mnemonic may be easier to remember for many programmers. Besides, use of the macro unburdens the programmer from remembering notational details, such as "\$+3."

Let's develop another example. It would be nice if the 8051 had instructions such as

```
JUMP  IF ACCUMULATOR GREATER THAN X  
JUMP  IF ACCUMULATOR GREATER THAN OR EQUAL TO X  
JUMP  IF ACCUMULATOR LESS THAN X  
JUMP  IF ACCUMULATOR LESS THAN OR EQUAL TO X
```

but it does not. These operations can be created using CJNE followed by JC or JNC, but the details are tricky. Suppose, for example, it is desired to jump to the label GREATER\_THAN if the accumulator contains an ASCII code greater than "Z" (5AH). The following instruction sequence would work:

```
CJNE  A, #5BH, $+3  
JNC   GREATER_THAN
```

The CJNE instruction subtracts 5BH (i.e., "Z" + 1) from the content of A and sets or clears the carry flag accordingly. CJNE leaves C=1 for accumulator values 00H up to and including 5AH. (Note: 5AH-5BH<0, therefore C=1; but 5BH-5BH=0, therefore C=0.) Jumping to GREATER\_THAN on the condition "not carry" correctly jumps for accumulator values 5BH, 5CH, 5DH, and so on, up to FFH. Once details such as these are worked out, they can be simplified by inventing an appropriate mnemonic, defining a macro, and using the macro instead of the corresponding instruction sequence. Here's the definition for a "jump if greater than" macro:

```
%*DEFINE      (JGT (VALUE, LABEL))  
              (CJNE  A, #%VALUE+1, $+3    ;JGT  
               JNC   %LABEL  
               )
```

To test if the accumulator contains an ASCII code greater than "Z," as just discussed, the macro would be called as

```
%JGT  ('Z', GREATER_THAN)
```

ASM51 would expand this into

```
CJNE  A, #5BH, $+3    ;JGT  
JNC   GREATER_THAN
```

The JGT macro is an excellent example of a relevant and powerful use of macros. By using macros, the programmer benefits by using a meaningful mnemonic and avoiding messy and potentially bug-ridden details.

---

## Local Labels

Local labels may be used within a macro using the following format:

```
%*DEFINE      (macro_name [(parameter_list)])
                [LOCAL list_of_local_labels] (macro_body)
```

For example, the following macro definition

```
%*DEFINE      (DEC_DPTR)  LOCAL SKIP
                (DEC      DPL                      ;DECREMENT DATA POINTER
                 MOV      A,      DPL
                 CJNE     A,      #0FFH, %SKIP
                 DEC      DPL
%SKIP:         )
```

would be called as

```
%DEC_DPTR
```

and would be expanded by ASM51 into

```
DEC      DPL                      ;DECREMENT DATA POINTER
MOV      A,      DPL
CJNE     A,      #0FFH, SKIP00
DEC      DPH
SKIP00:
```

Note that a local label generally will not conflict with the same label used elsewhere in the source program, since ASM51 appends a numeric code to the local label when the macro is expanded. Furthermore, the next use of the same local label receives the next numeric code, and so on.

The macro above has a potential "side effect." The accumulator is used as a temporary holding place for DPL. If the macro is used within a section of code that uses A for another purpose, the value in A would be lost. This side effect probably represents a bug in the program. The macro definition could guard against this by saving A on the stack. Here's an alternate definition for the DEC\_DPTR macro:

```
%*DEFINE      (DEC_DPTR)  LOCAL SKIP
                (PUSHACC
                 DEC      DPL                      ;DECREMENT DATA POINTER
                 MOV      A,      DPL
                 CJNE     A,      #0FFH, %SKIP
                 DEC      DPH
%SKIP:         POP      ACC
                )
```

## Repeat Operations

This is one of several built-in (predefined) macros. The format is

```
%REPEAT      (expression)      (text)
```

For example, to fill a block of memory with 100 NOP instructions,

```
%REPEAT      (100)
(NOP
)
```

---

## Control Flow Operations

The conditional assembly of section of code is provided by ASM51's control flow macro definition. The format is

```
%IF (expression) THEN (balanced_text)
[ELSE (balanced_text)] FI
```

For example,

```
INTRENAL      EQU      1          ;1 = 8051 SERIAL I/O DRIVERS
                                   ;0 = 8251 SERIAL I/O DRIVERS
                                   .
                                   .
                                   %IF (INTERNAL) THEN
(INCHAR:      .                  ;8051 DRIVERS
                                   .
OUTCHR:      .
                                   .
                                   ) ELSE
(INCHAR:      .                  ;8251 DRIVERS
                                   .
OUTCHR:      .
                                   .
                                   )
```

In this example, the symbol INTERNAL is given the value 1 to select I/O subroutines for the 8051's serial port, or the value 0 to select I/O subroutines for an external UART, in this case the 8251. The IF macro causes ASM51 to assemble one set of drivers and skip over the other. Elsewhere in the program, the INCHAR and OUTCHR subroutines are used without consideration for the particular hardware configuration. As long as the program is assembled with the correct value for INTERNAL, the correct subroutine is executed.

---

## Appendix B: 8051 C Programming

### ADVANTAGES AND DISADVANTAGES OF 8051 C

The advantages of programming the 8051 in C as compared to assembly are:

- Offers all the benefits of high-level, structured programming languages such as C, including the ease of writing subroutines
- Often relieves the programmer of the hardware details that the compiler handles on behalf of the programmer
- Easier to write, especially for large and complex programs
- Produces more readable program source codes

Nevertheless, 8051 C, being very similar to the conventional C language, also suffers from the following disadvantages:

- Processes the disadvantages of high-level, structured programming languages.
- Generally generates larger machine codes
- Programmer has less control and less ability to directly interact with hardware

To compare between 8051 C and assembly language, consider the solutions to the Example—Write a program using Timer 0 to create a 1KHz square wave on P1.0.

A solution written below in 8051 C language:

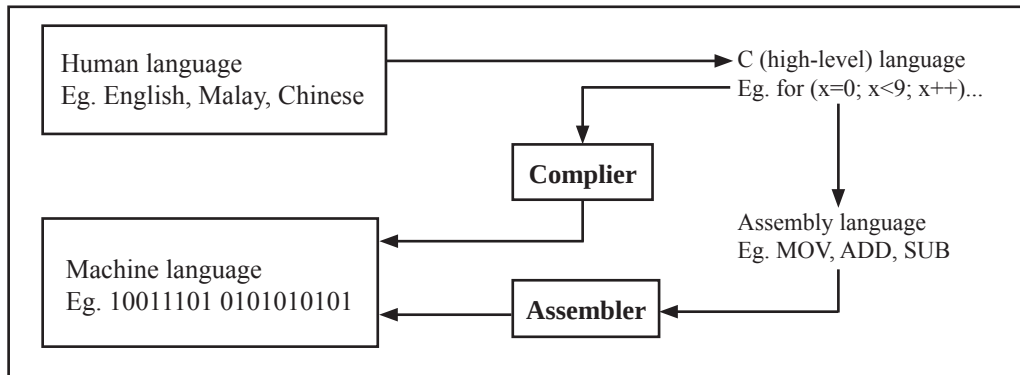
```
sbit portbit = P1^0;          /*Use variable portbit to refer to P1.0*/
main ()
{
    TMOD = 1;
    while (1)
    {
        TH0 = 0xFE;
        TL0 = 0xC;
        TR0 = 1;
        while (TF0 != 1);
        TR0 = 0;
        TF0 = 0;
        portbit = !(P1.^0);
    }
}
```

A solution written below in assembly language:

|       |      |            |                            |
|-------|------|------------|----------------------------|
|       | ORG  | 8100H      |                            |
|       | MOV  | TMOD, #01H | ;16-bit timer mode         |
| LOOP: | MOV  | TH0, #0FEH | ; -500 (high byte)         |
|       | MOV  | TL0, #0CH  | ; -500 (low byte)          |
|       | SETB | TR0        | ;start timer               |
| WAIT: | JNB  | TF0, WAIT  | ;wait for overflow         |
|       | CLR  | TR0        | ;stop timer                |
|       | CLR  | TF0        | ;clear timer overflow flag |
|       | CPL  | P1.0       | ;toggle port bit           |
|       | SJMP | LOOP       | ;repeat                    |
|       | END  |            |                            |

Notice that both the assembly and C language solutions for the above example require almost the same number of lines. However, the difference lies in the readability of these programs. The C version seems more human than assembly, and is hence more readable. This often helps facilitate the human programmer's efforts to write even very complex programs. The assembly language version is more closely related to the machine code, and though less readable, often results in more compact machine code. As with this example, the resultant machine code from the assembly version takes 83 bytes while that of the C version requires 149 bytes, an increase of 79.5%!

The human programmer's choice of either high-level C language or assembly language for talking to the 8051, whose language is machine language, presents an interesting picture, as shown in following figure.



Conversion between human, high-level, assembly, and machine language

## 8051 C COMPILERS

We saw in the above figure that a compiler is needed to convert programs written in 8051 C language into machine language, just as an assembler is needed in the case of programs written in assembly language. A compiler basically acts just like an assembler, except that it is more complex since the difference between C and machine language is far greater than that between assembly and machine language. Hence the compiler faces a greater task to bridge that difference.

Currently, there exist various 8051 C compiler, which offer almost similar functions. All our examples and programs have been compiled and tested with Keil's  $\mu$  Vision 2 IDE by Keil Software, an integrated 8051 program development environment that includes its C51 cross compiler for C. A cross compiler is a compiler that normally runs on a platform such as IBM compatible PCs but is meant to compile programs into codes to be run on other platforms such as the 8051.

## DATA TYPES

8051 C is very much like the conventional C language, except that several extensions and adaptations have been made to make it suitable for the 8051 programming environment. The first concern for the 8051 C programmer is the data types. Recall that a data type is something we use to store data. Readers will be familiar with the basic C data types such as `int`, `char`, and `float`, which are used to create variables to store integers, characters, or floating-points. In 8051 C, all the basic C data types are supported, plus a few additional data types meant to be used specifically with the 8051.

The following table gives a list of the common data types used in 8051 C. The ones in bold are the specific 8051 extensions. The data type **bit** can be used to declare variables that reside in the 8051's bit-addressable locations (namely byte locations 20H to 2FH or bit locations 00H to 7FH). Obviously, these bit variables can only store bit values of either 0 or 1. As an example, the following C statement:

```
bit flag = 0;
```

declares a bit variable called `flag` and initializes it to 0.

---

#### Data types used in 8051 C language

| Data Type      | Bits | Bytes | Value Range                                            |
|----------------|------|-------|--------------------------------------------------------|
| <b>bit</b>     | 1    |       | 0 to 1                                                 |
| signed char    | 8    | 1     | -128 to +127                                           |
| unsigned char  | 8    | 1     | 0 to 255                                               |
| enum           | 16   | 2     | -32768 to +32767                                       |
| signed short   | 16   | 2     | -32768 to +32767                                       |
| unsigned short | 16   | 2     | 0 to 65535                                             |
| signed int     | 16   | 2     | -32768 to +32767                                       |
| unsigned int   | 16   | 2     | 0 to 65535                                             |
| signed long    | 32   | 4     | -2,147,483,648 to +2,147,483,647                       |
| unsigned long  | 32   | 4     | 0 to 4,294,967,295                                     |
| float          | 32   | 4     | $\pm 1.175494\text{E-}38$ to $\pm 3.402823\text{E+}38$ |
| <b>sbit</b>    | 1    |       | 0 to 1                                                 |
| <b>sfr</b>     | 8    | 1     | 0 to 255                                               |
| <b>sfr16</b>   | 16   | 2     | 0 to 65535                                             |

The data type **sbit** is somewhat similar to the bit data type, except that it is normally used to declare 1-bit variables that reside in special function registers (SFRs). For example:

```
sbit    P = 0xD0;
```

declares the **sbit** variable P and specifies that it refers to bit address D0H, which is really the LSB of the PSW SFR. Notice the difference here in the usage of the assignment ("=") operator. In the context of **sbit** declarations, it indicates what address the **sbit** variable resides in, while in **bit** declarations, it is used to specify the initial value of the **bit** variable.

Besides directly assigning a bit address to an **sbit** variable, we could also use a previously defined **sfr** variable as the base address and assign our **sbit** variable to refer to a certain bit within that **sfr**. For example:

```
sfr      PSW = 0xD0;
sbit     P = PSW^0;
```

This declares an **sfr** variable called PSW that refers to the byte address D0H and then uses it as the base address to refer to its LSB (bit 0). This is then assigned to an **sbit** variable, P. For this purpose, the caret symbol (^) is used to specify bit position 0 of the PSW.

A third alternative uses a constant byte address as the base address within which a certain bit is referred. As an illustration, the previous two statements can be replaced with the following:

```
sbit     P = 0xD0 ^ 0;
```

Meanwhile, the **sfr** data type is used to declare byte (8-bit) variables that are associated with SFRs. The statement:

```
sfr      IE = 0xA8;
```

declares an **sfr** variable IE that resides at byte address A8H. Recall that this address is where the Interrupt Enable (IE) SFR is located; therefore, the **sfr** data type is just a means to enable us to assign names for SFRs so that it is easier to remember.

The **sfr16** data type is very similar to **sfr** but, while the **sfr** data type is used for 8-bit SFRs, **sfr16** is used for 16-bit SFRs. For example, the following statement:

```
sfr16    DPTR = 0x82;
```

declares a 16-bit variable DPTR whose lower-byte address is at 82H. Checking through the 8051 architecture, we find that this is the address of the DPL SFR, so again, the **sfr16** data type makes it easier for us to refer to the SFRs by name rather than address. There's just one thing left to mention. When declaring **sbit**, **sfr**, or **sfr16** variables, remember to do so outside main, otherwise you will get an error.

In actual fact though, all the SFRs in the 8051, including the individual flag, status, and control bits in the bit-addressable SFRs have already been declared in an include file, called reg51.h, which comes packaged with most 8051 C compilers. By using reg51.h, we can refer for instance to the interrupt enable register as simply IE rather than having to specify the address A8H, and to the data pointer as DPTR rather than 82H. All this makes 8051 C programs more human-readable and manageable. The contents of reg51.h are listed below.

```

/*-----
REG51.H
Header file for generic 8051 microcontroller.
-----*/

/* BYTE Register */
sfr    P0      = 0x80;
sfr    P1      = 0x90;
sfr    P2      = 0xA0;
sfr    P3      = 0xB0;
sfr    PSW     = 0xD0;
sfr    ACC     = 0xE0;
sfr    B       = 0xF0;
sfr    SP      = 0x81;
sfr    DPL     = 0x82;
sfr    DPH     = 0x83;
sfr    PCON    = 0x87;
sfr    TCON    = 0x88;
sfr    TMOD    = 0x89;
sfr    TL0     = 0x8A;
sfr    TL1     = 0x8B;
sfr    TH0     = 0x8C;
sfr    TH1     = 0x8D;
sfr    IE      = 0xA8;
sfr    IP      = 0xB8;
sfr    SCON    = 0x98;
sfr    SBUF    = 0x99;
/* BIT Register */
/* PSW */
sbit    CY      = 0xD7;
sbit    AC      = 0xD6;
sbit    F0      = 0xD5;
sbit    RS1     = 0xD4;
sbit    RS0     = 0xD3;
sbit    OV      = 0xD2;
sbit    P       = 0xD0;
/* TCON */
sbit    TF1     = 0x8F;
sbit    TR1     = 0x8E;
sbit    TF0     = 0x8D;
sbit    TR0     = 0x8C;

sbit    IE1     = 0x8B;
sbit    IT1     = 0x8A;
sbit    IE0     = 0x89;
sbit    IT0     = 0x88;
/* IE */
sbit    EA      = 0xAF;
sbit    ES      = 0xAC;
sbit    ET1     = 0xAB;
sbit    EX1     = 0xAA;
sbit    ET0     = 0xA9;
sbit    EX0     = 0xA8;
/* IP */
sbit    PS      = 0xBC;
sbit    PT1     = 0xBB;
sbit    PX1     = 0xBA;
sbit    PT0     = 0xB9;
sbit    PX0     = 0xB8;
/* P3 */
sbit    RD      = 0xB7;
sbit    WR      = 0xB6;
sbit    T1      = 0xB5;
sbit    T0      = 0xB4;
sbit    INT1    = 0xB3;
sbit    INT0    = 0xB2;
sbit    TXD     = 0xB1;
sbit    RXD     = 0xB0;
/* SCON */
sbit    SM0     = 0x9F;
sbit    SM1     = 0x9E;
sbit    SM2     = 0x9D;
sbit    REN     = 0x9C;
sbit    TB8     = 0x9B;
sbit    RB8     = 0x9A;
sbit    TI      = 0x99;
sbit    RI      = 0x98;

```

---

## MEMORY TYPES AND MODELS

The 8051 has various types of memory space, including internal and external code and data memory. When declaring variables, it is hence reasonable to wonder in which type of memory those variables would reside. For this purpose, several memory type specifiers are available for use, as shown in following table.

| Memory types used in 8051 C language |                                                         |
|--------------------------------------|---------------------------------------------------------|
| Memory Type                          | Description (Size)                                      |
| code                                 | Code memory (64 Kbytes)                                 |
| data                                 | Directly addressable internal data memory (128 bytes)   |
| idata                                | Indirectly addressable internal data memory (256 bytes) |
| bdata                                | Bit-addressable internal data memory (16 bytes)         |
| xdata                                | External data memory (64 Kbytes)                        |
| pdata                                | Paged external data memory (256 bytes)                  |

The first memory type specifier given in above table is **code**. This is used to specify that a variable is to reside in code memory, which has a range of up to 64 Kbytes. For example:

```
char    code    errmsg[ ] = "An error occurred" ;
```

declares a char array called errmsg that resides in code memory.

If you want to put a variable into data memory, then use either of the remaining five data memory specifiers in above table. Though the choice rests on you, bear in mind that each type of data memory affect the speed of access and the size of available data memory. For instance, consider the following declarations:

```
signed int  data  num1;
bit bdata   numbit;
unsigned int xdata num2;
```

The first statement creates a signed int variable num1 that resides in internal **data** memory (00H to 7FH). The next line declares a bit variable numbit that is to reside in the bit-addressable memory locations (byte addresses 20H to 2FH), also known as **bdata**. Finally, the last line declares an unsigned int variable called num2 that resides in external data memory, **xdata**. Having a variable located in the directly addressable internal data memory speeds up access considerably; hence, for programs that are time-critical, the variables should be of type **data**. For other variants such as 8052 with internal data memory up to 256 bytes, the **idata** specifier may be used. Note however that this is slower than data since it must use indirect addressing. Meanwhile, if you would rather have your variables reside in external memory, you have the choice of declaring them as **pdata** or **xdata**. A variable declared to be in **pdata** resides in the first 256 bytes (a page) of external memory, while if more storage is required, **xdata** should be used, which allows for accessing up to 64 Kbytes of external data memory.

What if when declaring a variable you forget to explicitly specify what type of memory it should reside in, or you wish that all variables are assigned a default memory type without having to specify them one by one? In this case, we make use of **memory models**. The following table lists the various memory models that you can use.

| Memory models used in 8051 C language |                                                                          |
|---------------------------------------|--------------------------------------------------------------------------|
| Memory Model                          | Description                                                              |
| Small                                 | Variables default to the internal data memory (data)                     |
| Compact                               | Variables default to the first 256 bytes of external data memory (pdata) |
| Large                                 | Variables default to external data memory (xdata)                        |

---

A program is explicitly selected to be in a certain memory model by using the C directive, `#pragma`. Otherwise, the default memory model is **small**. It is recommended that programs use the small memory model as it allows for the fastest possible access by defaulting all variables to reside in internal data memory.

The **compact** memory model causes all variables to default to the first page of external data memory while the **large** memory model causes all variables to default to the full external data memory range of up to 64 Kbytes.

## ARRAYS

Often, a group of variables used to store data of the same type need to be grouped together for better readability. For example, the ASCII table for decimal digits would be as shown below.

| ASCII table for decimal digits |                   |
|--------------------------------|-------------------|
| Decimal Digit                  | ASCII Code In Hex |
| 0                              | 30H               |
| 1                              | 31H               |
| 2                              | 32H               |
| 3                              | 33H               |
| 4                              | 34H               |
| 5                              | 35H               |
| 6                              | 36H               |
| 7                              | 37H               |
| 8                              | 38H               |
| 9                              | 39H               |

To store such a table in an 8051 C program, an array could be used. An array is a group of variables of the same data type, all of which could be accessed by using the name of the array along with an appropriate index.

The array to store the decimal ASCII table is:

```
int    table[10] =
{0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39};
```

Notice that all the elements of an array are separated by commas. To access an individual element, an index starting from 0 is used. For instance, `table[0]` refers to the first element while `table[9]` refers to the last element in this ASCII table.

## STRUCTURES

Sometime it is also desired that variables of different data types but which are related to each other in some way be grouped together. For example, the name, age, and date of birth of a person would be stored in different types of variables, but all refer to the person's personal details. In such a case, a structure can be declared. A structure is a group of related variables that could be of different data types. Such a structure is declared by:

```
struct    person {
            char name;
            int age;
            long DOB;
        };
```

Once such a structure has been declared, it can be used like a data type specifier to create structure variables that have the member's name, age, and DOB. For example:

```
struct    person    grace = {"Grace", 22, 01311980};
```

---

would create a structure variable `grace` to store the name, age, and data of birth of a person called Grace. Then in order to access the specific members within the person structure variable, use the variable name followed by the dot operator (.) and the member name. Therefore, `grace.name`, `grace.age`, `grace.DOB` would refer to Grace's name, age, and data of birth, respectively.

## POINTERS

When programming the 8051 in assembly, sometimes register such as R0, R1, and DPTR are used to store the addresses of some data in a certain memory location. When data is accessed via these registers, indirect addressing is used. In this case, we say that R0, R1, or DPTR are used to point to the data, so they are essentially pointers.

Correspondingly in C, indirect access of data can be done through specially defined pointer variables. Pointers are simply just special types of variables, but whereas normal variables are used to directly store data, pointer variables are used to store the addresses of the data. Just bear in mind that whether you use normal variables or pointer variables, you still get to access the data in the end. It is just whether you go directly to where it is stored and get the data, as in the case of normal variables, or first consult a directory to check the location of that data before going there to get it, as in the case of pointer variables.

Declaring a pointer follows the format:

```
data_type    *pointer_name;
```

where

|                           |                                                              |
|---------------------------|--------------------------------------------------------------|
| <code>data_type</code>    | refers to which type of data that the pointer is pointing to |
| <code>*</code>            | denotes that this is a pointer variable                      |
| <code>pointer_name</code> | is the name of the pointer                                   |

As an example, the following declarations:

```
int    * numPtr
int    num;
numPtr = &num;
```

first declares a pointer variable called `numPtr` that will be used to point to data of type `int`. The second declaration declares a normal variable and is put there for comparison. The third line assigns the address of the `num` variable to the `numPtr` pointer. The address of any variable can be obtained by using the address operator, `&`, as is used in this example. Bear in mind that once assigned, the `numPtr` pointer contains the address of the `num` variable, not the value of its data.

The above example could also be rewritten such that the pointer is straightaway initialized with an address when it is first declared:

```
int    num;
int    * numPtr = &num;
```

In order to further illustrate the difference between normal variables and pointer variables, consider the following, which is not a full C program but simply a fragment to illustrate our point:

```
int    num = 7;
int    * numPtr = &num;
printf ("%d\n", num);
printf ("%d\n", numPtr);
printf ("%d\n", &num);
printf ("%d\n", *numPtr);
```

---

The first line declare a normal variable, num, which is initialized to contain the data 7. Next, a pointer variable, numPtr, is declared, which is initialized to point to the address of num. The next four lines use the printf( ) function, which causes some data to be printed to some display terminal connected to the serial port. The first such line displays the contents of the num variable, which is in this case the value 7. The next displays the contents of the numPtr pointer, which is really some weird-looking number that is the address of the num variable. The third such line also displays the addresss of the num variable because the address operator is used to obtain num's address. The last line displays the actual data to which the numPtr pointer is pointing, which is 7. The \* symbol is called the indirection operator, and when used with a pointer, indirectly obtains the data whose address is pointed to by the pointer. Therefore, the output display on the terminal would show:

```
7
13452 (or some other weird-looking number)
13452 (or some other weird-looking number)
7
```

## A Pointer's Memory Type

Recall that pointers are also variables, so the question arises where they should be stored. When declaring pointers, we can specify different types of memory areas that these pointers should be in, for example:

```
int * xdata numPtr = &num;
```

This is the same as our previous pointer examples. We declare a pointer numPtr, which points to data of type int stored in the num variable. The difference here is the use of the memory type specifier **xdata** after the \*. This specifies that pointer numPtr should reside in external data memory (**xdata**), and we say that the pointer's memory type is **xdata**.

## Typed Pointers

We can go even further when declaring pointers. Consider the example:

```
int data * xdata numPtr = &num;
```

The above statement declares the same pointer numPtr to reside in external data memory (**xdata**), and this pointer points to data of type int that is itself stored in the variable num in internal data memory (**data**). The memory type specifier, **data**, before the \* specifies the *data memory type* while the memory type specifier, **xdata**, after the \* specifies the pointer memory type.

Pointer declarations where the data memory types are explicitly specified are called typed pointers. Typed pointers have the property that you specify in your code where the data pointed by pointers should reside. The size of typed pointers depends on the data memory type and could be one or two bytes.

## Untyped Pointers

When we do not explicitly state the data memory type when declaring pointers, we get untyped pointers, which are generic pointers that can point to data residing in any type of memory. Untyped pointers have the advantage that they can be used to point to any data independent of the type of memory in which the data is stored. All untyped pointers consist of 3 bytes, and are hence larger than typed pointers. Untyped pointers are also generally slower because the data memory type is not determined or known until the compiled program is run at runtime. The first byte of untyped pointers refers to the data memory type, which is simply a number according to the following table. The second and third bytes are, respectively, the higher-order and lower-order bytes of the address being pointed to.

An untyped pointer is declared just like normal C, where:

```
int * xdata numPtr = &num;
```

does not explicitly specify the memory type of the data pointed to by the pointer. In this case, we are using untyped pointers.

---

| Data memory type values stored in first byte of untyped pointers |                  |
|------------------------------------------------------------------|------------------|
| Value                                                            | Data Memory Type |
| 1                                                                | idata            |
| 2                                                                | xdata            |
| 3                                                                | pdata            |
| 4                                                                | data/bdata       |
| 5                                                                | code             |

## FUNCTIONS

In programming the 8051 in assembly, we learnt the advantages of using subroutines to group together common and frequently used instructions. The same concept appears in 8051 C, but instead of calling them subroutines, we call them **functions**. As in conventional C, a function must be declared and defined. A function definition includes a list of the number and types of inputs, and the type of the output (return type), plus a description of the internal contents, or what is to be done within that function.

The format of a typical function definition is as follows:

```
return_type  function_name (arguments)  [memory] [reentrant] [interrupt] [using]
{
    ...
}
```

where

|               |                                                               |
|---------------|---------------------------------------------------------------|
| return_type   | refers to the data type of the return (output) value          |
| function_name | is any name that you wish to call the function as             |
| arguments     | is the list of the type and number of input (argument) values |
| memory        | refers to an explicit memory model (small, compact or large)  |
| reentrant     | refers to whether the function is reentrant (recursive)       |
| interrupt     | indicates that the function is actually an ISR                |
| using         | explicitly specifies which register bank to use               |

Consider a typical example, a function to calculate the sum of two numbers:

```
int sum (int a, int b)
{
    return a + b;
}
```

This function is called `sum` and takes in two arguments, both of type `int`. The return type is also `int`, meaning that the output (return value) would be an `int`. Within the body of the function, delimited by braces, we see that the return value is basically the sum of the two arguments. In our example above, we omitted explicitly specifying the options: `memory`, `reentrant`, `interrupt`, and `using`. This means that the arguments passed to the function would be using the default small memory model, meaning that they would be stored in internal data memory. This function is also by default non-recursive and a normal function, not an ISR. Meanwhile, the default register bank is bank 0.

## Parameter Passing

In 8051 C, parameters are passed to and from functions and used as function arguments (inputs). Nevertheless, the technical details of where and how these parameters are stored are transparent to the programmer, who does not need to worry about these technicalities. In 8051 C, parameters are passed through the register or through memory. Passing parameters through registers is faster and is the default way in which things are done. The registers used and their purpose are described in more detail below.

---

| Registers used in parameter passing |                       |                      |            |                 |
|-------------------------------------|-----------------------|----------------------|------------|-----------------|
| Number of Argument                  | Char / 1-Byte Pointer | INT / 2-Byte Pointer | Long/Float | Generic Pointer |
| 1                                   | R7                    | R6 & R7              | R4–R7      | R1–R3           |
| 2                                   | R5                    | R4 & R5              | R4–R7      |                 |
| 3                                   | R3                    | R2 & R3              |            |                 |

Since there are only eight registers in the 8051, there may be situations where we do not have enough registers for parameter passing. When this happens, the remaining parameters can be passed through fixed memory locations. To specify that all parameters will be passed via memory, the NOREGPARMs control directive is used. To specify the reverse, use the REGPARMs control directive.

## Return Values

Unlike parameters, which can be passed by using either registers or memory locations, output values must be returned from functions via registers. The following table shows the registers used in returning different types of values from functions.

| Registers used in returning values from functions |                |                                         |
|---------------------------------------------------|----------------|-----------------------------------------|
| Return Type                                       | Register       | Description                             |
| bit                                               | Carry Flag (C) |                                         |
| char/unsigned char/1-byte pointer                 | R7             |                                         |
| int/unsigned int/2-byte pointer                   | R6 & R7        | MSB in R6, LSB in R7                    |
| long/unsigned long                                | R4–R7          | MSB in R4, LSB in R7                    |
| float                                             | R4–R7          | 32-bit IEEE format                      |
| generic pointer                                   | R1–R3          | Memory type in R3, MSB in R2, LSB in R1 |

## Appendix C: STC15F204EA series MCU Electrical Characteristics

### Absolute Maximum Ratings

| Parameter                 | Symbol    | Min  | Max       | Unit |
|---------------------------|-----------|------|-----------|------|
| Storage temperature       | TST       | -55  | +125      | °C   |
| Operating temperature (I) | TA        | -40  | +85       | °C   |
| Operating temperature (C) | TA        | 0    | +70       | °C   |
| DC power supply (5V)      | VDD - VSS | -0.3 | +5.5      | V    |
| DC power supply (3V)      | VDD - VSS | -0.3 | +3.6      | V    |
| Voltage on any pin        | -         | -0.3 | VCC + 0.3 | V    |

### DC Specification (5V MCU)

| Sym  | Parameter                                                                    | Specification |       |      |      | Test Condition |
|------|------------------------------------------------------------------------------|---------------|-------|------|------|----------------|
|      |                                                                              | Min.          | Typ   | Max. | Unit |                |
| VDD  | Operating Voltage                                                            | 3.3           | 5.0   | 5.5  | V    |                |
| IPD  | Power Down Current                                                           | -             | < 0.1 | -    | uA   | 5V             |
| IIDL | Idle Current                                                                 | -             | 3.0   | -    | mA   | 5V             |
| ICC  | Operating Current                                                            | -             | 4     | 20   | mA   | 5V             |
| VIL1 | Input Low (P0,P1,P2,P3)                                                      | -             | -     | 0.8  | V    | 5V             |
| VIH1 | Input High (P0,P1,P2,P3)                                                     | 2.0           | -     | -    | V    | 5V             |
| VIH2 | Input High (RESET)                                                           | 2.2           | -     | -    | V    | 5V             |
| IOL1 | Sink Current for output low (P0,P1,P2,P3)                                    | -             | 20    | -    | mA   | 5V@Vpin=0.45V  |
| IOH1 | Sourcing Current for output high (P0,P1,P2,P3)<br>(Quasi-output)             | 200           | 270   | -    | uA   | 5V             |
| IOH2 | Sourcing Current for output high (P0,P1,P2,P3)<br>(Push-Pull, Strong-output) | -             | 20    | -    | mA   | 5V@Vpin=2.4V   |
| IIL  | Logic 0 input current (P0,P1,P2,P3)                                          | -             | -     | 50   | uA   | Vpin=0V        |
| ITL  | Logic 1 to 0 transition current (P0,P1,P2,P3)                                | 100           | 270   | 600  | uA   | Vpin=2.0V      |

### DC Specification (3V MCU)

| Sym  | Parameter                                                        | Specification |      |      |      | Test Condition  |
|------|------------------------------------------------------------------|---------------|------|------|------|-----------------|
|      |                                                                  | Min.          | Typ  | Max. | Unit |                 |
| VDD  | Operating Voltage                                                | 2.4           | 3.3  | 3.6  | V    |                 |
| IPD  | Power Down Current                                               | -             | <0.1 | -    | uA   | 3.3V            |
| IIDL | Idle Current                                                     | -             | 2.0  | -    | mA   | 3.3V            |
| ICC  | Operating Current                                                | -             | 4    | 10   | mA   | 3.3V            |
| VIL1 | Input Low (P0,P1,P2,P3)                                          | -             | -    | 0.8  | V    | 3.3V            |
| VIH1 | Input High (P0,P1,P2,P3)                                         | 2.0           | -    | -    | V    | 3.3V            |
| VIH2 | Input High (RESET)                                               | 2.2           | -    | -    | V    | 3.3V            |
| IOL1 | Sink Current for output low (P0,P1,P2,P3)                        | -             | 20   | -    | mA   | 3.3V@Vpin=0.45V |
| IOH1 | Sourcing Current for output high (P0,P1,P2,P3)<br>(Quasi-output) | 140           | 170  | -    | uA   | 3.3V            |
| IOH2 | Sourcing Current for output high (P0,P1,P2,P3)<br>(Push-Pull)    | -             | 20   | -    | mA   | 3.3V            |
| IIL  | Logic 0 input current (P0,P1,P2,P3)                              | -             | 8    | 50   | uA   | Vpin=0V         |
| ITL  | Logic 1 to 0 transition current (P0,P1,P2,P3)                    | -             | 110  | 600  | uA   | Vpin=2.0V       |

---

## Appendix D: STC15F204EA series to replace standard 8051 Notes

STC15F204EA series MCU Timer0/Timer1 is fully compatible with the traditional 8051 MCU. After power on reset, the default input clock source is the divider 12 of system clock frequency. STC15Fxx MCU instruction execution speed is faster than the traditional 8051 MCU 8 ~ 12 times in the same working environment, so software delay programs need to be adjusted.

### ALE

Traditional 8051's ALE pin output signal on divide 6 the system clock frequency can be externally provided clock, while STC15Fxx series MCU has no ALE pin, you can get clock source from CLKOUT1/P3.4, CLKOUT0/P3.5 or SYSCLK(P0.0 pin).

ALE pin is an disturbance source when traditional 8051's system clock frequency is too high. STC89xx series MCU add ALEOFFF bit in AUXR register. While STC15Fxx series MCU has no ALE pin and can remove ALE disturbance thoroughly. Please compare the following two registers.

### AUXR register of STC89xx series

| Mnemonic | Add | Name                 | Bit7 | Bit6 | Bit5 | Bit4 | Bit3 | Bit2 | Bit1   | Bit0   | Reset Value |
|----------|-----|----------------------|------|------|------|------|------|------|--------|--------|-------------|
| AUXR     | 8EH | Auxiliary register 0 | -    | -    | -    | -    | -    | -    | EXTRAM | ALEOFF | xxxx,xx00   |

### AUXR register of STC15F204EA series

| Mnemonic | Add | Name               | Bit7  | Bit6  | Bit5 | Bit4 | Bit3 | Bit2 | Bit1 | Bit0 | Reset Value |
|----------|-----|--------------------|-------|-------|------|------|------|------|------|------|-------------|
| AUXR     | 8EH | Auxiliary register | T0x12 | T1x12 | -    | -    | -    | -    | -    | -    | 00xx,xxxx   |

### PSEN

Traditional 8051 execute external program through the PSEN signal, STC15F204EA series have no PSEN signal.

### General Qusi-Bidirectional I/O

Traditional 8051 access I/O (signal transition or read status) timing is 12 clocks, STC15F204EA series MCU is 4 clocks. When you need to read an external signal, if internal output a rising edge signal, for the traditional 8051, this process is 12 clocks, you can read at once, but for STC15F204EA series MCU, this process is 4 clocks, when internal instructions is complete but external signal is not ready, so you must delay 1~2 nop operation.

### Port drive capability

STC15F204EA series I/O port sink drive current is 20mA, has a strong drive capability, the port is not burn out when drive high current generally. STC89 series I/O port sink drive current is only 6mA, is not enough to drive high current. For the high current drive applications, it is strongly recommended to use STC15F204EA series MCU.

### WatchDog

STC15F204EA series MCU's watch dog timer control register (WDT\_CONTR) is location at C1H, add watch dog reset flag.

### STC15F204EA series WDT\_CONTR ( C1H )

| Mnemonic  | Add | Name                            | Bit7     | Bit6 | Bit5   | Bit4    | Bit3    | Bit2 | Bit1 | Bit0 | Reset Value |
|-----------|-----|---------------------------------|----------|------|--------|---------|---------|------|------|------|-------------|
| WDT_CONTR | C1h | Wact-Dog-Timer Control register | WDT_FLAG | -    | EN_WDT | CLR_WDT | IDL_WDT | PS2  | PS1  | PS0  | xx00,0000   |

## STC89 series WDT\_CONTR ( E1H )

| Mnemonic  | Add | Name                            | Bit7 | Bit6 | Bit5   | Bit4    | Bit3    | Bit2 | Bit1 | Bit0 | Reset Value |
|-----------|-----|---------------------------------|------|------|--------|---------|---------|------|------|------|-------------|
| WDT_CONTR | E1h | Wact-Dog-Timer Control register | -    | -    | EN_WDT | CLR_WDT | IDL_WDT | PS2  | PS1  | PS0  | xx00,0000   |

STC15F204EA series MCU auto enable watch dog timer after ISP upgrade, but not in STC89 series, so STC15F204EA series's watch dog is more reliable.

## EEPROM

SFR associated with EEPROM

| Mnemonic  | STC15Fxx | STC89xx | Description                         |
|-----------|----------|---------|-------------------------------------|
|           | Address  |         |                                     |
| IAP_DATA  | C2H      | E2H     | ISP/IAP Flash data register         |
| IAP_ADDRH | C3H      | E3G     | ISP/IAP Flash HIGH address register |
| IAP_ADDRL | C4H      | E4H     | ISP/IAP Flash LOW address register  |
| IAP_CMD   | C5H      | E5H     | ISP/IAP Flash command register      |
| IAP_TRIG  | C6H      | E6H     | ISP/IAP command trigger register    |
| IAP_CONTR | C7H      | E7H     | ISP/IAP control register            |

STC15F204EA series write 5AH and A5H sequential to trigger EEPROM flash command, and STC89 series write 46H and B9H sequential to trigger EEPROM flash command.

STC15F204EA series EEPROM start address all location at 0000H, but STC89 series is not.

## Power consumption

Power consumption consists of two parts: crystal oscillator amplifier circuits and digital circuits. STC15F204EA series have no crystal oscillator amplifier circuits, so its consumption is lower than STC89 series. For digital circuits, the higher clock frequency, the greater the power consumption. STC15F204EA series MCU instruction execution speed is faster than the STC89 series MCU 3~24 times in the same working environment, so if you need to achieve the same efficiency, STC15F204EA series required frequency is lower than STC89 series MCU.

## PowerDown Wakeup

STC15F204EA series MCU wake-up support for rising edge or falling edge depend on the external interrupt mode, but STC89 series only support for low level.

## About reset circuit

For STC89 series, if the system frequency is below 12MHz, the external reset circuit is not required. Reset pin can be connected to ground through the 1K resistor or can be connected directly to ground. The proposal to create PCB to retain RC reset circuit.

While STC15F204EA series has an internal high-reliability reset circuit and does not need external reset circuit.

## About Clock oscillator

For STC89 series, if you need to use internal RC oscillator, XTAL1 pin and XTAL2 pin must be floating. If you use a external active crystal oscillator, clock signal input from XTAL1 pin and XTAL2 pin floating.

While STC15F204EA series only has an high-precision RC oscillator with temperature drifting  $\pm 1\%$  and has removed expensive external crystal oscillator.

## About power

Power at both ends need to add a 47uF electrolytic capacitor and a 0.1uF capacitor, to remove the coupling and filtering

## Appendix E: STC15F204EA series Selection Table

| Type<br>1T 8051 MCU | Operating<br>voltage<br>(V) | Flash<br>(B) | SRAM<br>(B) | TIMER | A/D    | WDT | EEPROM<br>(B) | Internal<br>low<br>voltage<br>interrupt | Internal<br>Reset<br>threshold<br>voltage<br>can be<br>configured | External<br>interrupts<br>which can<br>wake up<br>power down<br>mode | Special<br>timer for<br>waking<br>power<br>down<br>mode | Package of 28-pin<br>(26 I/O ports)<br>Price (RMB ¥) |          |
|---------------------|-----------------------------|--------------|-------------|-------|--------|-----|---------------|-----------------------------------------|-------------------------------------------------------------------|----------------------------------------------------------------------|---------------------------------------------------------|------------------------------------------------------|----------|
|                     |                             |              |             |       |        |     |               |                                         |                                                                   |                                                                      |                                                         | SOP-28                                               | SKDIP-28 |
| STC15F201A          | 5.5~3.8                     | 1K           | 256         | 2     | 10-bit | Y   | -             | Y                                       | Y                                                                 | 5                                                                    | N                                                       |                                                      |          |
| STC15F201EA         | 5.5~3.8                     | 1K           | 256         | 2     | 10-bit | Y   | 2K            | Y                                       | Y                                                                 | 5                                                                    | N                                                       | ¥                                                    | ¥        |
| STC15F202A          | 5.5~3.8                     | 2K           | 256         | 2     | 10-bit | Y   | -             | Y                                       | Y                                                                 | 5                                                                    | N                                                       |                                                      |          |
| STC15F202EA         | 5.5~3.8                     | 2K           | 256         | 2     | 10-bit | Y   | 2K            | Y                                       | Y                                                                 | 5                                                                    | N                                                       | ¥                                                    | ¥        |
| STC15F203A          | 5.5~3.8                     | 3K           | 256         | 2     | 10-bit | Y   | -             | Y                                       | Y                                                                 | 5                                                                    | N                                                       |                                                      |          |
| STC15F203EA         | 5.5~3.8                     | 3K           | 256         | 2     | 10-bit | Y   | 2K            | Y                                       | Y                                                                 | 5                                                                    | N                                                       | ¥                                                    | ¥        |
| STC15F204A          | 5.5~3.8                     | 4K           | 256         | 2     | 10-bit | Y   | -             | Y                                       | Y                                                                 | 5                                                                    | N                                                       |                                                      |          |
| STC15F204EA         | 5.5~3.8                     | 4K           | 256         | 2     | 10-bit | Y   | 1K            | Y                                       | Y                                                                 | 5                                                                    | N                                                       | ¥                                                    | ¥        |
| STC15F205A          | 5.5~3.8                     | 5K           | 256         | 2     | 10-bit | Y   | -             | Y                                       | Y                                                                 | 5                                                                    | N                                                       |                                                      |          |
| STC15F205EA         | 5.5~3.8                     | 5K           | 256         | 2     | 10-bit | Y   | 1K            | Y                                       | Y                                                                 | 5                                                                    | N                                                       | ¥                                                    | ¥        |
| IAP15F206A          | 5.5~3.8                     | 6K           | 256         | 2     | 10-bit | Y   | IAP           | Y                                       | Y                                                                 | 5                                                                    | N                                                       |                                                      |          |
| Type<br>1T 8051 MCU | Operating<br>voltage<br>(V) | Flash<br>(B) | SRAM<br>(B) | TIMER | A/D    | WDT | EEPROM<br>(B) | Internal<br>low<br>voltage<br>interrupt | Internal<br>Reset<br>threshold<br>voltage<br>can be<br>configured | External<br>interrupts<br>which can<br>wake up<br>power down<br>mode | Special<br>timer for<br>waking<br>power<br>down<br>mode | Package of 28-pin<br>(26 I/O ports)<br>Price (RMB ¥) |          |
|                     |                             |              |             |       |        |     |               |                                         |                                                                   |                                                                      |                                                         | SOP-28                                               | SKDIP-28 |
| STC15L201A          | 3.6~2.4                     | 1K           | 256         | 2     | 10-bit | Y   | -             | Y                                       | Y                                                                 | 5                                                                    | N                                                       |                                                      |          |
| STC15L201EA         | 3.6~2.4                     | 1K           | 256         | 2     | 10-bit | Y   | 2K            | Y                                       | Y                                                                 | 5                                                                    | N                                                       | ¥                                                    | ¥        |
| STC15L202A          | 3.6~2.4                     | 2K           | 256         | 2     | 10-bit | Y   | -             | Y                                       | Y                                                                 | 5                                                                    | N                                                       |                                                      |          |
| STC15L202EA         | 3.6~2.4                     | 2K           | 256         | 2     | 10-bit | Y   | 2K            | Y                                       | Y                                                                 | 5                                                                    | N                                                       | ¥                                                    | ¥        |
| STC15L203A          | 3.6~2.4                     | 3K           | 256         | 2     | 10-bit | Y   | -             | Y                                       | Y                                                                 | 5                                                                    | N                                                       |                                                      |          |
| STC15L203EA         | 3.6~2.4                     | 3K           | 256         | 2     | 10-bit | Y   | 2K            | Y                                       | Y                                                                 | 5                                                                    | N                                                       | ¥                                                    | ¥        |
| STC15L204A          | 3.6~2.4                     | 4K           | 256         | 2     | 10-bit | Y   | -             | Y                                       | Y                                                                 | 5                                                                    | N                                                       |                                                      |          |
| STC15L204EA         | 3.6~2.4                     | 4K           | 256         | 2     | 10-bit | Y   | 1K            | Y                                       | Y                                                                 | 5                                                                    | N                                                       | ¥                                                    | ¥        |
| STC15L205A          | 3.6~2.4                     | 5K           | 256         | 2     | 10-bit | Y   | -             | Y                                       | Y                                                                 | 5                                                                    | N                                                       |                                                      |          |
| STC15L205EA         | 3.6~2.4                     | 5K           | 256         | 2     | 10-bit | Y   | 1K            | Y                                       | Y                                                                 | 5                                                                    | N                                                       | ¥                                                    | ¥        |
| IAP15L206A          | 3.6~2.4                     | 6K           | 256         | 2     | 10-bit | Y   | IAP           | Y                                       | Y                                                                 | 5                                                                    | N                                                       |                                                      |          |