C51 Family Programmer's Guide and Instruction Set

Summary

1. Memory Organization

1.1. Program Memory

The TEMIC C51 Microcontroller Family has separate address spaces for program Memory and Data Memory. The program memory can be up to 64 K bytes long. The lower 4 K for the 80C51 (8 K for the 80C52, 16 K for the 83 C154 and 32 K for the 83C154D) may reside on chip. Figure 1 to 4 show a map of 80C51, 80C52, 83C154 and 83C154D program memory.

Figure 1. The 80C51 Program Memory.

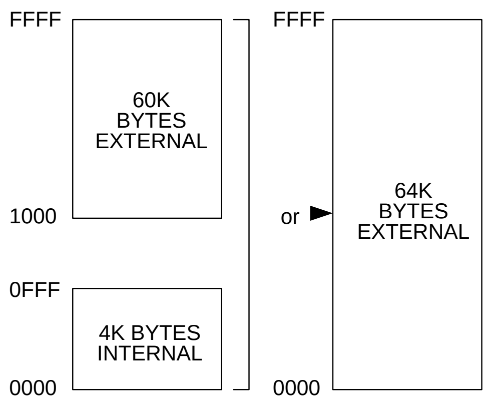


Figure 2. The 80C52 Program Memory.

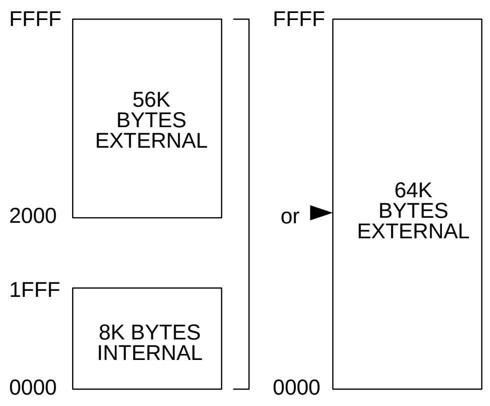


Figure 3. The 83C154 Program Memory.

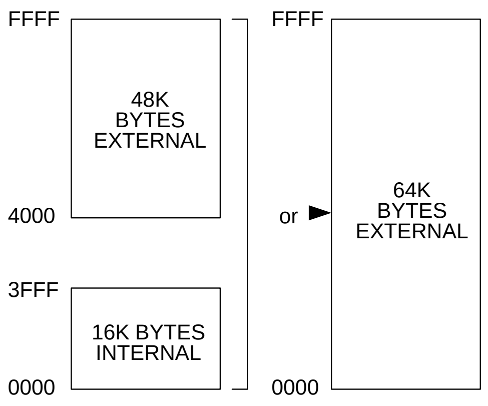
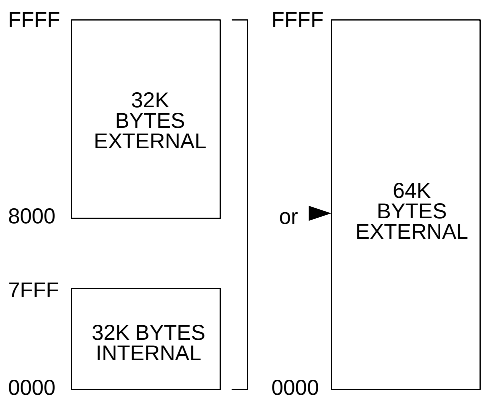


Figure 4. The 83C154D Program Memory.



1.2. Data Memory

The C51 Microcontroller Family can address up to 64 K bytes of Data Memory to the chip. The “MOVX” instruction is used to access the external data memory (refer to the C51 instruction set, in this chapter, for detailed description of instructions).

The 80C51 has 128 bytes of on-chip-RAM (256 bytes in the 80C52, 83C154 and 83C154D) plus a number of Special Function Registers (SFR). The lower 128 bytes of RAM can be accessed either by direct addressing (MOV data addr). or by indirect addressing (MOV @Ri). Figure 5 and 6 show the 80C51, 80C52, 83C154 and 83C154D Data Memory organization.

Figure 5. The 80C51 Data Memory Organisation.

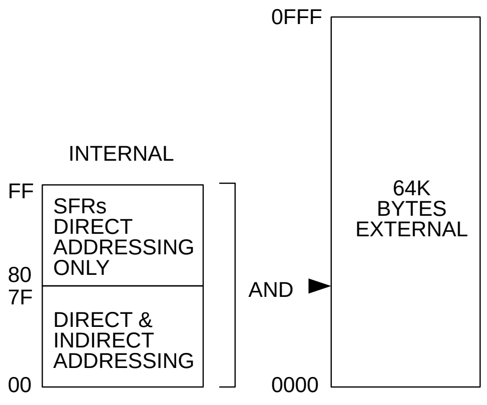
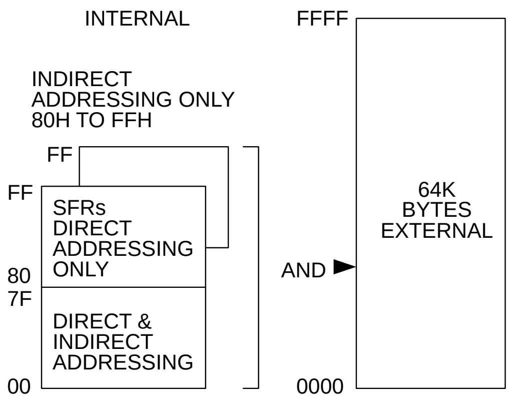


Figure 6. The 80C52, 83C154 and 83C154D Data Memory Organisation.


INTERNAL
FFFF
INDIRECT ADDRESSING ONLY 80H TO FFH
FF
FF
SFRs DIRECT ADDRESSING ONLY
80
7F
DIRECT & INDIRECT ADDRESSING
00
AND
64K BYTES EXTERNAL
0000

1.3. Indirect Address Area

Note that in Figure 6 - the SFRs and the indirect address RAM have the same addresses (80H-OFFH). Nevertheless, they are two separate areas and are accessed in two different ways. For example the instruction

```
MOV 80H, #0AAH
```

writes 0AAH to Port 0 which is one of the SFRs and the instruction

```
MOV R0, # 80H
MOV @ R0, # 0BBH
```

writes 0BBH in location 80H of the data RAM. Thus, after execution of both of the above instructions Port 0 will contain 0AAH and location 80 of the RAM will contain 0BBH.

Note that the stack operations are examples of indirect addressing, so the upper 128 bytes of data RAM are available as stack space in those devices which implement 256 bytes of internal RAM.

1.4. Direct And Indirect Address Area

The 128 bytes of RAM which can be accessed by both direct and indirect addressing can be divided into 3 segments as listed below and shown in figure 7.

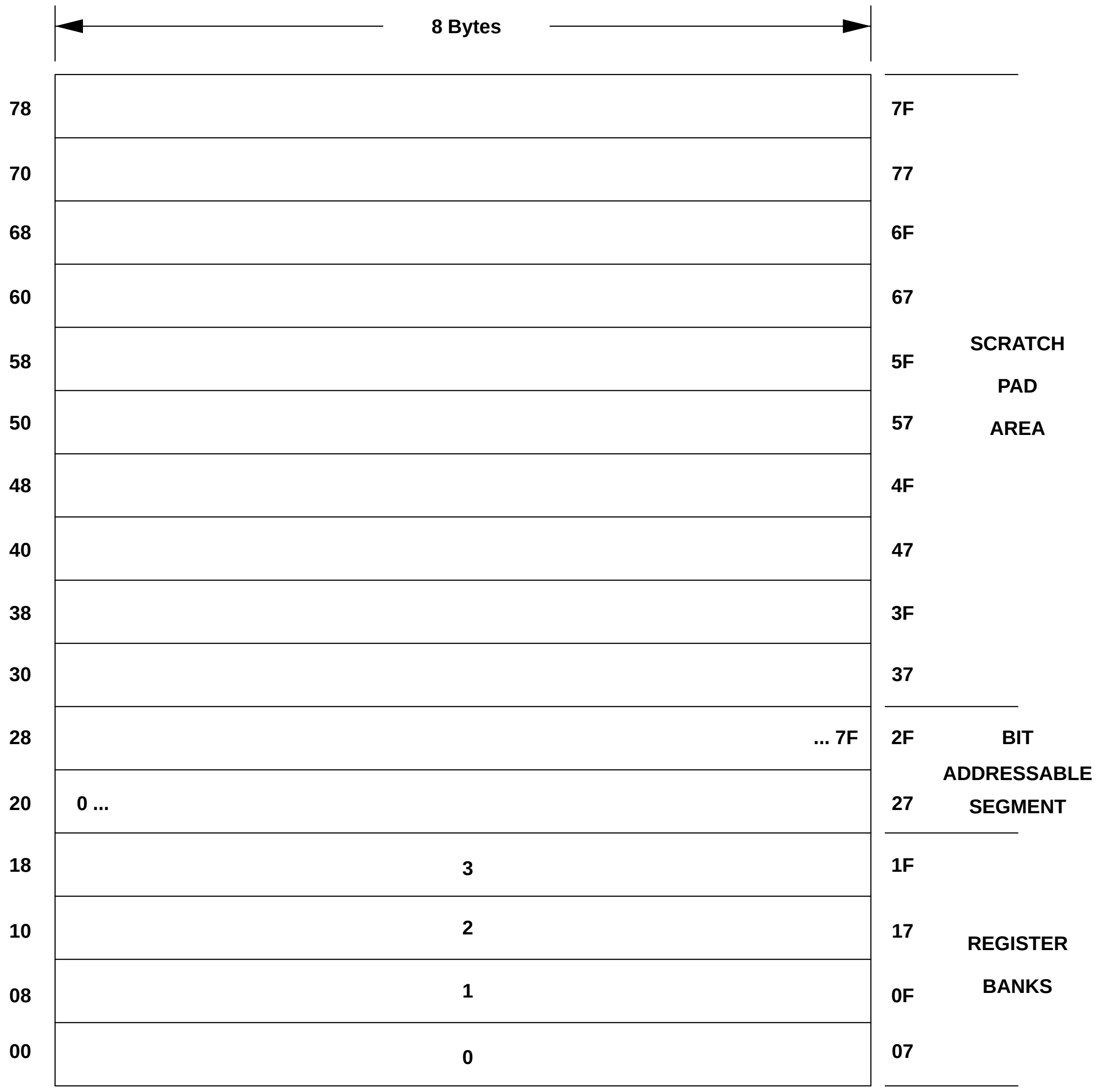
1. Register Banks 0.3 : Locations 0 through 1FH (32 bytes). ASM-51 and the device after reset default to register bank 0. To use the other register banks the user must select them in the software. Each register bank contains 8 one-byte registers, 0 through 7.
Reset initializes the Stack Pointer to location 07H and it is incremented once to start from location 08H which is the first register (R0) of the second register bank. Thus, in order to use more than one register bank, the SP should be initialized to a different location of the RAM where it is not used for data storage (ie, higher part of the RAM).

2. Bit Addressable Area : 16 bytes have been assigned for this segment, 20H-2FH. Each one of the 128 bits of this segment can be directly addressed (0-7FH).
The bits can be referred to in two ways both of which are

acceptable by the ASM-51. One way is to refer to their addresses, ie, 0 to 7FH. The other way is with reference to bytes 20H to 2FH. Thus, bits 0-7 can also be referred to as bits 20.0-20.7, and bits 8-FH are the same as 21.0-21.7 and so on.
Each of the 16 bytes in this segment can also be addresses as a byte.

3. Scratch Pad Area : Bytes 30H through 7FH are available to user as data RAM. However, if the stack pointer has been initialized to this area, enough number of bytes should be left aside to prevent SP data destruction.

Figure 7. 128 Bytes of RAM Direct and Indirect Addressable.



1.5. Special Function Registers

Table 1 contains a list of all the SFRs and their addresses. Comparing table 1 and figure 8 shows that all of the SFRs that are byte and bit addressable are located on the first column of the diagram in figure 8.

Table 1.

SYMBOL	NAME	ADDRESS
*ACC	Accumulator	0E0H
*B	B Register	0F0H
*PSW	Program Status Word	0D0H
SP	Stack Pointer	81H
DPTR	Data Pointer 2 Bytes	
DPL	Low Byte	82H
DPH	High Byte	83H
*P0	Port 0	80H
*P1	Port 1	90H
*P2	Port 2	0A0H
*P3	Port 3	0B0H
*IP	Interrupt Priority Control	0B8H
*IE	Interrupt Enable Control	0A8H
TMOD	Timer/Counter Mode Control	89H
*TCON	Timer/Counter Control	88H
*+T2CON	Timer/Counter 2 Control	0C8H
TH0	Timer/Counter 0 High Byte	8CH
TL0	Timer/Counter 0 Low Byte	8AH
TH1	Timer/Counter 1 High Byte	8DH
TL1	Timer/Counter 1 Low Byte	8BH
+TH2	Timer/Counter 1 High Byte	0CDH
+TL2	Timer/Counter 2 Low Byte	0CCH
+RCAP2H	T/C 2 Capture Reg. High Byte	0CBH
+RCAP2L	T/C 2 Capture Reg. Low Byte	0CAH
*SCON	Serial Control	98H
SBUF	Serial Data Buffer	99H
PCON	Power Control	87H
*IOCON (1)	IO Control	F8H

+ 80C52, 83C154 and 83C154D only * bit addressable
(1) 83C154 and 83C154D only

2. SFR Memory Map

Figure 8.

8 Bytes

F8	IOCON								FF
F0	B								F7
E8									EF
E0	ACC								E7
D8									DF
D0	PSW								D7
C8	T2CON		RCAP2L	RCAP2H	TL2	TH2			CF
C0									C7
B8	IP								BF
B0	P3								B7
A8	IE								AF
A0	P2								A7
98	SCON	SBUF							9F
90	P1								97
88	TCON	TMOD	TL0	TL1	TH0	TH1			8F
80	P0	SP	DPL	DPH				PCON	87

↑ bit addressable

2.1. What do the SFRs Contain just after Power-on or a Reset ?

Table 2 lists the contents of each SFR after a power-on reset or a hardware reset.

Table 2. Contents of the SRFs after reset.

REGISTER	VALUE IN BINARY
*ACC	0000 0000
*B	0000 0000
*PSW	0000 0000
SP	0000 0111
DPTR	0000 0000
*P0	1111 1111
*P1	1111 1111
*P2	1111 1111
*P3	1111 1111
*IP	XXX0 0000 80C51
	XXX0 0000 80C52
	0X00 0000 83C154/C154D
*IE	0XX0 0000 80C51
	0X000 0000 83C154/C154D and 80C52
TMOD	0000 0000

* : bit addressable.
+ : 80C52, 83C154 and 83C154D only.
– : 83C154 and 83C154D only.
X : Undefined.

REGISTER	VALUE IN BINARY
*TCON	0000 0000
+*T2CON	0000 0000
TH0	0000 0000
TL0	0000 0000
TH1	0000 0000
TL1	0000 0000
+ TH2	0000 0000
+ TL2	0000 0000
+ RCAP2L	0000 0000
+ RCAP2H	0000 0000
*SCON	0000 0000
SBUF	Indeterminate
PCON	0XXX 0000 80C51 and 80C52 000X 0000 83C154 and 83C154D
–*IOCON	0000 0000

These SFRs that have their bits assigned for various functions are listed in this section. A brief description of each bit is provided for quick reference. For more detailed information refer to the Architecture chapter of this book.

PSW : Program Status Word (Bit Addressable)

CY	AC	F0	RS1	RS0	OV	F1	P

CY	PSW.7	Carry Flag.
AC	PSW.6	Auxiliary Carry Flag.
F0	PSW.5	Flag 0 available to the user for general purpose.
RS1	PSW.4	Register Bank selector bit 1 (SEE NOTE).
RS0	PSW.3	Register Bank selector bit 0 (SEE NOTE).
OV	PSW.2	Overflow Flag.
F1	PSW.1	Flag F1 available to the user for general purpose.
P	PSW.0	Parity flag. Set/cleared by hardware each instruction cycle to indicate an odd/even number of “1” bits in the accumulator.

Note :

The value presented by RS0 and RS1 selects the corresponding register bank.

RS1	RS0	REGISTER BANK	ADDRESS
0	0	0	00H–07H
0	1	1	08H–0FH
1	0	2	10H–17H
1	1	3	18H–1FH

* User software should not write 1s to reserved bits. These bits may be used in future TEMIC C51 products to invoke new features. In that case, the reset or inactive value of the new bit will be 0, and its active value will be 1.

PCON : Power Control Register (Not Bit Addressable)

SMOD	HPD	RPD	–	GF1	GF0	PD	IDL

SMOD	PCON.7	Double baud rate bit. If SMOD = 1, the baud rate is doubled when the serial part is used in mode 1, 2 and 3.
HPD	PCON.6	Hard Power Down. (83C154 and 83C154D only). The falling/rising edge of a signal connected on pin P3.5 Starts/Stops the Power-Down mode. A reset can also stop this mode.
RPD	PCON.5	Recover Power Down bit. (83C154 and 83C154D only). It's used to cancel a Power-Down/IDLE mode. If it's set, an interrupt (enable or disable) can cancel this mode. A reset can also stop this mode (see Note 1).
-	PCON.4	Not implemented, reserved for futur used*
GF1	PCON.3	General purpose bit.
GF0	PCON.2	General purpose bit.
PD	PCON.1	Power Down bit. If set, the oscillator is stopped. A reset or an interrupt (83C154 and 83C154D only) can cancel this mode (Note 1).
IDL	PCON.0	IDLE bit. If set the activity CPU is stopped. A reset or an interrupt can cancel this mode (See Note 1).

* User software should not write 1s to reserved bits. These bits may be used in future TEMIC C51 products to invoke new features. In that case, the reset or inactive value of the new bit will be 0, and its active value will be 1.

Note 1 (83C154 and 83C154D only) :

– if RPD = 0 and if an interrupt cancels the mode Power-Down/IDLE, the next instruction to execute is a LCALL at the interrupt routine.

– RPD = 1 – if interrupt request is enable the next instruction to execute is a LCALL at the interrupt routine.
– if interrupt request is disable, the program continue with the instruction immediately after the Power-Down/Idle instruction.

2.2. Interrupts

In order to use any of the interrupts in the C51, the following three steps must be taken.
1. Set the EA (enable all) bit in the IE register to 1.
2. Set the corresponding individual interrupt enable bit in the IE register to 1.
3. Begin the Interrupt service routine at the corresponding Vector Address of that interrupt. See Table below.

INTERRUPT SOURCE	VECTOR ADDRESS
IE0	0003H
TF0	000BH
IE1	0013H
TF1	001BH
RI & TI	0023H
TF2 & EXF2	002BH

In addition, for external interrupts, pins $\overline{\text{INT0}}$ and $\overline{\text{INT1}}$ (P3.2 and P3.3) must be set to 1, and depending on whether the interrupt is to be level or transition activated, bits IT0 or IT1 in the TCON register may need to be set to 1.

ITX = 0 level activated
ITX = 1 transition activated

IE : Interrupt Enable Register (Bit Addressable)

If the bit is 0, the corresponding interrupt is disabled. If the bit is 1, the corresponding interrupt is enabled.

EA	–	ET2	ES	ET1	EX1	ET0	EX0

EA	IE.7	Disables all interrupts. If EA = 0, no interrupt will be acknowledged. If EA = 1, interrupt source is individually enable or disabled by setting or clearing its enable bit.
-	IE.6	Not implemented, reserved for future use*.
ET2	IE.5	Enable or disable the Timer 2 overflow or capture interrupt (80C52, 83C154 and 83C154D only).
ES	IE.4	Enable or disable the Serial port interrupt.
ET1	IE.3	Enable or disable the Timer 1 overflow interrupt.
EX1	IE.2	Enable or disable External interrupt 1.
ET0	IE.1	Enable or disable the Timer 0 overflow interrupt.
EX0	IE.0	Enable or disable External Interrupt 0.

* User software should not write 1s to reserved bits. These bits may be used in future TEMIC C51 products to invoke new features. In that case, the reset or inactive value of the new bit will be 0, and its active value will be 1.

2.3. Assigning Higher Priority to one More Interrupts

In order to assign higher priority to an interrupt the corresponding bit in the IP register must be set to 1. Remember that while an interrupt service is in progress, it cannot be interrupted by a lower or same level interrupt.

2.4. Priority Within Level

Priority within level is only to resolve simultaneous requests of the same priority level. From high to low, interrupt sources are listed below :

IE0
TF0
IE1
TF1
RI or TI
TF2 or EXF2

IP : Interrupt Priority Register (Bit Addressable)

If the bit is 0, the corresponding interrupt has a lower priority and if the bit is the corresponding interrupt has a higher priority.

PCT	–	PT2	PS	PT1	PX1	PT0	PX0

PCT	IP.7	Defines the same priority level for all the source interrupt (83C154 and 83C154D only).
-	IP.6	Not implemented, reserved for future use*.
PT2	IP.5	Defines the Timer 2 interrupt priority level (80C52, 83C154 and 83C154D only).
PS	IP.4	Defines the Serial Port interrupt priority level.
PT1	IP.3	Defines the Timer 1 Interrupt priority level.
PX1	IP.2	Defines External Interrupt priority level.
PT0	IP.1	Defines the Timer 0 interrupt priority level.
PX0	IP.0	Defines the External Interrupt 0 priority level.

* User software should not write 1s to reserved bits. These bits may be used in future TEMIC C51 products to invoke new features. In that case, the reset or inactive value of the now bit will be 0, and its active value will be 1.

IOCON : Input/Output Control Register (83C154 and 83C154D only)

WDT	T32	SERR	IZC	P3HZ	P2HZ	P1HZ	ALF

WDT	IOCON.7	Watch Dog Timer bit. Set when Timer 1 is overflow (TF = 1). The CPU is reset and the program is executed from address 0.
T32	IOCON.6	Timer 32 bits. The Timer 1 and Timer 0 are connected together to form a 32 bits Timer/Counter. If C/TO = 0, it's a Timer. If C/TO = 1, it's a counter.
SERR	IOCON.5	Serial Port Reception Error flag. Set when an overrun on frame error is received.
IZC	IOCON.4	Set/Cleared by software to select 100/10 K pull up resistance for Port 1, 2 and 3.
P3HZ	IOCON.3	When Set, Port 3 becomes a tri-state input. When cleared, the pull-up resistance value is selected by IZC.
P2HZ	IOCON.2	When Set, Port 2 becomes a tri-state input. When cleared, the pull-up resistance value is selected by IZC.
P1HZ	IOCON.1	When Set, Port 1 becomes a tri-state input. When cleared, the pull-up resistance value is selected by IZC.
ALF	IOCON.0	All Port tri-state. When Set and CPU in Power-Down mode, port 1, 2 and 3 are tri-state.

TCON : Timer/Counter Control Register (Bit Addressable)

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1	TCON.7	Timer 1 overflow flag. Set by hardware when the Timer/Counter 1 overflows. Cleared by hardware as processor vectors to the interrupt service routine.
TR1	TCON.6	Timer 1 run control bit. Set/cleared by software to turn Timer/Counter ON/OFF.
TF0	TCON.5	Timer 0 overflow flag. Set by hardware when the Timer/Counter 0 overflows. Cleared by hardware as processor vectors to the service routine.
TR0	TCON.4	Timer 0 run control bit. Set/cleared by software to turn Timer/Counter 0 ON/OFF.
IE1	TCON.3	External Interrupt 1 edge flag. Set by hardware when External interrupt edge is detected. Cleared by hardware when interrupt is processed.
IT1	TCON.2	Interrupt 1 type control bit. Set/cleared by software to specify falling edge/flow level triggered External Interrupt.
IE0	TCON.1	External Interrupt 0 edge flag. Set by hardware when External Interrupt edge detected. Cleared by hardware when interrupt is processed.
IT0	TCON.0	Interrupt 0 type control bit. Set/cleared by software to specify falling edge/low level triggered External Interrupt.

TMOD : Timer/Counter Mode Control Register (Not Bit Addressable)

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

GATE	When TRx (in TCON) is set and GATE = 1, TIMER/COUNTERx will run only while INTx pin is high (hardware control). When GATE = 0, TIMER/COUNTERx will run only while TRx = 1 (software control).
C/$\overline{T}$	Timer or Counter selector. Cleared for Timer operation (input from internal system clock). Set for Counter operation (input from Tx input pin).
M1	Mode selector bit (NOTE 1).
M0	Mode selector bit (NOTE 1).

Note 1 :

M1	M0	OPERATING MODE	
0	0	0	13–bit Timer
0	1	1	16–bit Timer/Counter
1	0	2	8–bit Auto–Reload Timer/Counter
1	1	3	(Timer 0) TL0 is an 8–bit Timer/Counter controlled by the standard Timer 0 control bits, TH0 is an 8–bit Timer and is controlled by Timer 1 control bits.
1	1	3	(Timer 1) Timer/Counter 1 stopped.

2.5. Timer Set-up

Tables 3 through 6 give some values for TMOD which can be used to set up Timer 0 in different modes.

It is assumed that only one timer is being used at a time. It is desired to run Timers 0 and 1 simultaneously, in any mode, the value that in TMOD for Timer 0 must be ORed with the value shown for Timer 1 (Tables 5 and 6).

For example, if it is desired to run Timer 0 in mode 1 GATE (external control) and Timer 1 in mode 2 COUNTER, then the value must be loaded into TMOD is 69H (09H from Table 3 ORed with 60H from Table 6).

Moreover, it is assumed that the user, at this point, is not ready to turn the timers on and will do that a different point in the program by setting bit TRx (in TCON) to 1.

Table 4. As a Counter

MODE	TIMER 0 FUNCTION	TMOD	
		INTERNAL CONTROL (NOTE 1)	EXTERNAL CONTROL (NOTE 2)
0	13–bit Timer	04H	0CH
1	16–bit Timer	05H	0DH
2	8–bit Auto–Reload	06H	0EH
3	one 8–bit Timers	07H	0FH

2.6. Timer/Counter 0

Table 3. As a Timer

MODE	TIMER 0 FUNCTION	TMOD	
		INTERNAL CONTROL (NOTE 1)	EXTERNAL CONTROL (NOTE 2)
0	13–bit Timer	00H	08H
1	16–bit Timer	01H	09H
2	8–bit Auto–Reload	02H	0AH
3	Two 8–bit Timers	03H	0BH

Notes : 1. The Timer is turned ON/OFF by setting/clearing bit TR0 in the software.
2. The Timer is turned ON/OFF by the 1 to 0 transition on $\overline{\text{INT0}}$ (P3.2) when TR0 = 1 (hardware control).

2.7. Timer/Counter 1

Table 5. As a Timer

MODE	TIMER 0 FUNCTION	TMOD	
		INTERNAL CONTROL (NOTE 1)	EXTERNAL CONTROL (NOTE 2)
0	13–bit Timer	00H	80H
1	16–bit Timer	10H	90H
2	8–bit Auto–Reload	20H	A0H
3	does not run	30H	B0H

Table 6. As a Counter

MODE	TIMER 0 FUNCTION	TMOD	
		INTERNAL CONTROL (NOTE 1)	EXTERNAL CONTROL (NOTE 2)
0	13–bit Timer	40H	C0H
1	16–bit Timer	50H	D0H
2	8–bit Auto–Reload	60H	E0H
3	not available	–	–

Notes : 1. The Timer is turned ON/OFF by setting/clearing bit TR1 in the software.
2. The Timer is turned ON/OFF by the 1 to 0 transition on $\overline{\text{INT1}}$ (P3.2) when TR1 = 1 (hardware control).

T2CON : Timer/Counter 2 Control register (Bit Addressable) (80C52, 83C154 and 83C154D only)

TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/T2	CP/RL2

TF2	T2CON.7	Timer 2 overflow flag set by hardware and cleared by software. TF2 cannot be set when either RCLK = 1 or CLK = 1
EXF2	T2CON.6	Timer 2 external flag set when either a capture or reload is caused by a negative transition on T2EX, and EXEN2 = 1. When Timer 2 interrupt is enabled, EXF2 = 1 will cause the CPU to vector to the Timer 2 interrupt routine. EXF2 must be cleared by software.
RCLK	T2CON.5	Receive clock flag. When set, causes the Serial Port to use Timer 2 overflow pulses for its receive clock in modes 1 & 3. RCLK = 0 causes Timer 1 overflow to be used for the receive clock.
TCLK	T2CON.4	Transmit clock flag. When set, causes the Serial Port use Timer 2 overflow pulses for its transmit clock in modes 1 & 3, TCLK = 0 causes Timer 1 overflows to be used for the transmit clock.
EXEN2	T2CON.3	Timer 2 external enable flag. When set, allows a capture or reload to occur as a result of negative transition on T2EX if Timer 2 is not being used to clock the Serial Port. EXEN2 = 0 causes Timer 2 to ignore events as T2EX.
TR2	T2CON.2	Software START/STOP control for Timer 2. A logic 1 starts the Timer.
C/$\overline{\text{T2}}$	T2CON.1	Timer or Counter select.
CP/$\overline{\text{RL2}}$	T2CON.0	Capture/Reload flag. When set, captures will occur on negative transitions at T2EX if EXEN2 = 1. When cleared, Auto-Reloads will occur either with Timer2 overflows or negative transitions at T2EX when EXEN2 = 1. When either RCLK = 1 or TCLK = 1, this bit is ignored and the Timer is forced to Auto-Reload on Timer 2 overflow.

2.8. Timer/Counter 2 Set-up

Except for the baud rate generator mode, the values given for T2CON do not include the setting of the TR2 bit. Therefore, bit TR2 must be set, separately, to turn the Timer on.

Table 7. As a Timer

MODE	T2CON	
	INTERNAL CONTROL (NOTE 1)	EXTERNAL CONTROL (NOTE 2)
16–bit Auto–Reload	00H	08H
16–bit Capture	01H	09H
BAUD rate generator		
receive & transmit same		
baud rate	34H	36H
receive only	24H	26H
transmit only	14H	16H

Table 8. As a Counter

MODE	T2CON	
	INTERNAL CONTROL (NOTE 1)	EXTERNAL CONTROL (NOTE 2)
16–bit Auto–Reload	02H	0AH
16–bit Capture	03H	0BH

Notes : 1. Capture/Reload occurs only Timer/Counter overflow.
2. Capture/Reload occurs on Timer/Counter overflow and a 1 to 0 transition on T2EX (P1.1) pin except when Timer 2 is used in the baud rate generating mode.

SCON : Serial Port Control Register (Bit Addressable)

SM0	SM1	SM2	REN	TN8	RB8	TI	RI

SM0	SCON.7	Serial Port mode specifier (NOTE 1).
SM1	SCON.6	Serial Port mode specifier (NOTE 1).
SM2	SCON.5	Enables the multiprocessor communication feature in mode 2 & 3. In mode 2 or 3, if SM2 is set to 1 then RI will not be activated if the received 9th data bit (RB8) is 0. In mode 1, if SM2 = 1 then RI will not be activated if a valid stop bit was not received. In mode 0, SM2 should be 0 (See table 9).
REN	SCON.4	Set/Cleared by software to Enable/Disable reception.
TB8	SCON.3	The 9th bit that will be transmitted in modes 2 & 3. Set/Cleared by software.
RB8	SCON.2	In modes 2 & 3, is the 9th data bit that was received. In mode 1, if SM2 = 0, RB8 is the stop bit that was received. In mode 0, RB8 is not used.
TI	SCON.1	Transmit interrupt flag. Set by hardware at the end of the 8th bit time in mode 0, or at the beginning of the stop bit in the other modes. Must be cleared by software.
RI	SCON.0	Receive interrupt flag. Set by hardware at the end of the 8th bit time in mode 0, or half way through the stop bit time in the other modes (except see SM2). Must be cleared by software.

Note 1 :

SM0	SM1	MODE	DESCRIPTION	BAUD RATE
0	0	0	SHIFT REGISTER	Fosc./12
0	1	1	8 bit UART	Variable
1	0	2	8 bit UART	Fosc./64 OR Fosc./32
1	1	3	8 bit UART	Variable

2.9. Serial Port Set-Up

Table 9.

MODE	SCON	SM2 VARIATION
0	10H	
1	50H	Single Processor
2	90H	Environment
3	D0H	(SM2 = 0)
0	NA	
1	70H	Multiprocessor
2	B0H	Environment
3	FOH	(SM2 = 1)

2.10. Generating Baud Rates

Serial Port in Mode 0 :
Mode 0 has a fixed baud rate which is 1/12 of oscillator frequency. To run serial port in this mode none of the Timer/Counters need to be set up. Only the SCON register needs to be defined.

$$\text{Baud Rate} = \frac{\text{Osc Freq}}{12}$$

Serial Port in Mode 1 :
Mode 1 has a variable baud rate. The baud rate can be generated by either Timer 1 or Timer 2 (80C52, 83C154 and 83C154D only).

2.11. Using Timer/Counter 1 to Generate Baud Rates

For this purpose, Timer 1 is used in mode 2 (Auto-Reload). Refer to Timer Setup section of this chapter.

$$\text{Baud Rate} = \frac{K \times \text{Oscillator freq.}}{32 \times 12 \times [256–(TH1)]}$$

if SMOD = 0, then K = 1.

If SMOD = 1, then K = 2. (SMOD is the PCON register). Most of the time the user knows the baud rate and needs to know the reload value for TH1. Therefore, the equation to calculate TH1 can be written as :

$$TH1 = 256 - \frac{K \times \text{Oscillator freq.}}{384 \times \text{baud rate}}$$

TH1 must be integer value. Rounding off TH1 to the nearest integer may not produce the desired baud rate. In this case, the user may have to choose another crystal frequency.

Since the PCON register is not bit addressable, one way to set the bit is logical ORing the PCON register (ie, ORL PCON, #80H). The address of PCON is 87H.

2.12. Using Timer/Counter 2 to Generate Baud Rates

For this purpose, Timer 2 must be used in the baud rate generating mode. Refer to Timer 2 Setup Table in this chapter. If Timer 2 is being clocked through pin T2 (P1.0) the baud rate is :

$$\text{Baud Rate} = \frac{\text{Timer 2 Overflow Rate}}{16}$$

And if it being clocked internally the baud rate is :

$$\text{Baud Rate} = \frac{\text{Osc. Freq}}{32 \times [65536 - (\text{RCAP2H, RCAP2L})]}$$

To obtain the reload value for RCAP2H and RCAP2L the above equation can be written as :

$$\text{RCAP2H, RCAP2L} = 65536 - \frac{\text{Osc. Freq}}{32 \times \text{Baud rate}}$$

2.13. Serial Port in Mode 2

The baud rate is fixed in this mode and 1/32 or 1/64 of the oscillator frequency depending on the value of the SMOD bit in the PCON register.

In this mode none of the Timers are used and the clock comes from the internal phase 2 clock.

SMOD = 1, Baud Rate = 1/32 Osc Freq.

SMOD = 0, Baud Rate = 1/64 Osc Freq.

To set the SMOD bit : ORL PCON, #80H. The address of PCON is 87H.

2.14. Serial Port in Mode 3

The baud rate in mode 3 is variable and sets up exactly the same as in mode 1.

Table 10. TEMIC C51 Instruction Set

Interrupt Response time : Refer to Hardware Description Chapter.

Instructions that Affect Flag Settings (1)

INSTRUC.	FLAG			INSTRUC.	FLAG		
	C	OV	AC		C	OV	AC
ADD	X	X	X	CLRC	O		
ADDC	X	X	X	CPL C	X		
SUBB	X	X	X	ANL C, bit	X		
MUL	0	X		ANL C,/ bit	X		
DIV	0	X		ORL C, bit		X	
DA	X			ORL C, bit		X	
RRC	X			MOV C, bit		X	
RLC	X			CJNE	X		
SETB C	1						

(1) note that operations on SFR byte address 208 or bit addresses 209–215 (i.e., the PSW or bits in the PSW) will also affect flag settings.

Note on instruction set and addressing modes :

Rn	– Register R7–R0 of the currently selected Register Bank
direct	– 8-bit internal data location’s address. This could be an Internal Data RAM location (0–127) or a SFR (i.e., I/O port, control register, status register, etc. (128–255)).
@Ri	– 8-bit internal data RAM location (0-255) addresses indirectly through register R1 or R0.
# data	– 8-bit constant included in instruction.
# data 16	– 16-bit constant included in instruction.
addr 16	– 16-bit destination address. Used by LCALL & LJMP. Abranch can be anywhere within the 64K-byte Program memory address space
addr 11	– 11-bit destination address. Used by ACALL & AJMP. The branch will be within the same 2K–byte page of program memory as the first byte of the following instruction
rel	– Signed (two’s complement) 8-bit offset byte. Used by SJMP and all conditionnal jumps. Range is –128 to + 127 bytes relative to first byte of the following instruction.
bit	– Direct Addressed bit in internal Data RAM or special Function Register.

MNEMONIC	DESCRIPTION	BYTE	OSCIL. PERIOD
ARITHMETIC OPERATIONS			
ADD A, Rn	Add register to Accumulator	1	12
ADD A, direct	Add direct byte to Accumulator	2	12
ADD A, @Ri	Add indirect RAM to Accumulator	1	12
ADD A, #data	Add immediate data to Accumulator	2	12
ADDCA, Rn	Add register to Accumulator with Carry	1	12
ADDCA, direct	Add direct byte to Accumulator with Carry	2	12
ADDCA, @Ri	Add indirect RAM to Accumulator with Carry	1	12
ADDCA, #data	Add immediate data to Acc with Carry	2	12
SUBB A, Rn	Subtract Register from Acc with borrow	1	12
SUBB A, direct	Subtract direct byte from Acc with borrow	2	12
SUBB A, @Ri	Subtract indirect RAM from ACC with borrow	1	12
SUBB A, #data	Subtract immediate data from Acc with borrow	2	12
INC A	Increment Accumulator	1	12
INC Rn	Increment register	1	12
INC direct	Increment direct byte	2	12
INC @Ri	Increment direct RAM	1	12
DEC A	Decrement Accumulator	1	12
DEC Rn	Decrement Register	1	12
DEC direct	Decrement direct byte	2	12
DEC @Ri	Decrement indirect RAM	1	12
INC DPTR	Increment Data Pointer	1	24
MUL AB	Multiply A & B	1	48
DIV AB	Divide A by B	1	48
DA A	Decimal Adjust Accumulator	1	12

MNEMONIC	DESCRIPTION	BYTE	OSCIL. PERIOD
LOGICAL OPERATIONS			
ANL A, Rn	AND Register to Accumulator	1	12
ANL A, direct	AND direct byte to Accumulator	2	12
ANL A, @Ri	AND indirect RAM to Accumulator	1	12
ANL A, #data	AND immediate data to Accumulator	2	12
ANL direct, A	AND Accumulator to direct byte	2	12
ANL direct, #data	AND immediate data to direct byte	3	24
ORL A, Rn	OR register to Accumulator	1	12
ORL A, direct	OR direct byte to Accumulator	2	12
ORL A, @Ri	OR indirect RAM to Accumulator	1	12
ORL A, #data	OR immediate data to Accumulator	2	12
ORL direct, A	OR Accumulator to direct byte	2	12
ORL direct, #data	OR immediate data to direct byte	3	24
XRL A, Rn	Exclusive-OR register to Accumulator	1	12
XRL A, direct	Exclusive-OR direct byte to accumulator	2	12
XRL A, @Ri	Exclusive-OR indirect RAM to Accumulator	1	12
XRL A, #data	Exclusive–OR immediate data to Accumulator	2	12
XRL direct, A	Exclusive–OR Accumulator to direct byte	2	12
XRL direct, #data	Exclusive–OR immediate data to direct byte	3	24
CLR A	Clear Accumulator	1	12
CPL A	Complement Accumulator	1	12
RL A	Rotate Accumulator Left	1	12
RLC A	Rotate Accumulator Left through the Carry	1	12
RR A	Rotate Accumulator Right	1	12
RRC A	Rotate Accumulator Right through the Carry	1	12
SWAP A	Swap nibbles within the Accumulator	1	12

MNEMONIC	DESCRIPTION	BYTE	OSCIL. PERIOD
DATA TRANSFERT			
MOV A, Rn	Move Register to Accumulator	1	12
MOV A, direct	Move direct byte to Accumulator	2	12
MOV A, @Ri	Move indirect RAM to Accumulator	1	12
MOV A, #data	Move immediate data to Accumulator	2	12
MOV Rn, A	Move Accumulator to register	1	12
MOV Rn, direct	Move direct byte to register	2	24
MOV Rn, #data	Move immediate data to register	2	12
MOV direct, A	Move Accumulator to direct byte	2	12
MOV direct, Rn	Move register to direct byte	2	24
MOV direct, direct	Move direct byte to direct	3	24
MOV direct, @Ri	Move indirect RAM to direct byte	2	24
MOV direct, #data	Move immediate data to direct byte	3	24
MOV @Ri, A	Move Accumulator to indirect RAM	1	12
MOV @Ri, direct	Move direct by to indirect RAM	2	24
MOV @Ri, #data	Move immediate data to indirect RAM	2	12
MOV DPTR, #data16	Load Data Pointer with a 16–bit constant	3	24
MOVCA @A+DPTR	Move Code byte relative to DPTR to Acc	1	24
MOVCA @A+PC	Move Code byte relative to PC to Acc	1	24
MOVXA, @Ri	Move External RAM (8-bit addr) to Acc	1	24
MOVXA, @DPTR	Move External RAM (16-bit addr) to Acc	1	24
MOVX@Ri, A	Move Acc to External RAM (8-bit addr)	1	24
MOVX@DPTR, A	Move Acc to External RAM (16-bit addr)	1	24
PUSH direct	Push direct byte only stack	2	24
POP direct	Pop direct byte from stack	2	24

MNEMONIC	DESCRIPTION	BYTE	OSCIL. PERIOD
DATA TRANSFERT (continued)			
XCH A, Rn	Exchange register with Accumulator	1	12
XCH A, direct	Exchange direct byte with Accumulator	2	12
XCH A, @Ri	Exchange indirect RAM with Accumulator	1	12
XCHD A, @Ri	Exchange loworder Digit indirect RAM with Acc	1	12
BOOLEAN VARIABLE MANIPULATION			
CLR C	Clear Carry	1	12
CLR bit	Clear direct bit	2	12
SETB C	Set Carry	1	12
SETB bit	Set direct bit	2	12
CPL C	Complement Carry	1	12
CPL bit	Complement direct bit	2	12
ANL C, bit	AND direct bit to Carry	2	24
ANL C, /bit	AND complement of direct bit to Carry	2	24
ORL C, bit	OR direct bit to Carry	2	24
ORL C, /bit	OR complement of direct bit to Carry	2	24
MOV C, bit	Move direct bit to Carry	2	12
MOV bit, C	Move Carry to direct bit	2	24
JC rel	Jump if Carry is set	2	24
JNC rel	Jump if Carry not set	2	24
JB bit, rel	Jump if direct Bit is set	3	24
JNB bit, rel	Jump if direct Bit is Not set	3	24
JBC bit, rel	Jump if direct Bit is set & clear bit	3	24

MNEMONIC	DESCRIPTION	BYTE	OSCIL. PERIOD
PROGRAM BRANCHING			
ACALLK addr11	Absolute Subroutine Call	2	24
LCALL addr16	Long Subroutine Call	3	24
RET	Return from Subroutine	1	24
RETI	Return from interrupt	1	24
AJMPaddr11	Absolute Jump	2	24
LJMPaddr16	Long Jump	3	24
SJMP rel	Short Jump (relative addr)	2	24
JMP @A+DPTR	Jump direct relative to the DPTR	1	24
JZ rel	Jump if Accumulator is zero	2	24
JNZ rel	Jump if Accumulator is not Zero	2	24
CNJE A, direct, rel	Compare direct byte to Acc and Jump if Not Equal	3	24
CJNE A, #data, rel	Compare immediate to Acc and Jump if Not Equal	3	24
CJNE Rn, #data, rel	Compare immediate to register and Jump if Not Equal	3	24
CJNE @Ri, #data, rel	Compare immediate to indirect and Jump if Not Equal	3	24
DJNZ Rn, rel	Decrement register and Jump if Not Zero	2	24
DJNZ direct, rel	Decrement direct byte and Jump if Not Zero	3	24
NOP	No Operation	1	12

3. Instruction Definitions

ACALL addr 11

Function : Absolute Call

Description : ACALL unconditionally calls a subroutine located at the indicated address. The instruction increments the PC twice to obtain the address of the following instruction, then pushes the 16-bit result onto the stack (low-order byte first) and increments the Stack Pointer twice. The destination address is obtained by 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 2 K block of the program memory as the first byte of the instruction following ACALL. No flags are affected.

Example : Initially SP equals 07H. The labs " SUBRTN " is at program memory location 0345 H. 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	0	1	a7	a6	a5	a4	a3	a2	a1	a0

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

ADD a, <src-byte>

Function : Add

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

OV is set 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

Byte : 1

Cycle : 1

Encoding :

0	0	1	0	1	r	r	r

Operation : ADD
(A) ← (A) + (Rn)

ADD A, direct

Bytes : 2

Cycle : 1

Encoding :

0	0	1	0	0	1	0	1	direct address

Operation : ADD
(A) ← (A) + (direct)

ADD A, @RI

Byte : 1

Cycle : 1

Encoding :

0	0	1	0	1	1	1	i

Operation : ADD
(A) ← (A) + ((RI))

ADD A, # data

Bytes : 2

Cycle : 1

Encoding :

0	0	1	0	0	1	0	0	Immediate data

Operation : ADD
(A) ← (A) + # data

ADDC A, <src-byte>

Function : Add with Carry

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

OV is set if there is a carry-out of bit 6 but not out of bit 7, or a carry-out of bit 7 but not out of bit 6 ; otherwise OV is cleared. When adding signed intergers, 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 mode are allowed ; register, direct, register-indirect, or immediate.

Example : The Accumulator holds 0C3H (11000011B) and register 0 holds 0AAH (10101010B) with the carry flag set. The instruction,

ADDC A, R0

will leave 6EH (01101110B) in the Accumulator with AC cleared and both the Carry flag and OV set to 1.

ADDC A, RN

Byte : 1

Cycle : 1

Encoding :

0	0	1	1	1	r	r	r

Operation : ADDC
$(A) \leftarrow (A) + (C) + (R_n)$

ADDC A, direct

Bytes : 2

Cycle : 1

Encoding :

0	0	1	1	0	1	0	1	direct address

Operation : ADDC
$(A) \leftarrow (A) + (C) + (\text{direct})$

ADDC A, @ RI

Byte : 1

Cycle : 1

Encoding :

0	0	1	1	0	1	1	i

Operation : ADDC
$(A) \leftarrow (A) + (C) + ((R_i))$

ADDC A, #data

Bytes : 2

Cycle : 1

Encoding :

0	0	1	1	0	1	0	0	immediate data

Operation : ADDC
$(A) \leftarrow (A) + (C) + \#\ \text{data}$

AJMP addr11

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 2 K 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 a location 0345H and will load the PC with 0123H.

ADD A, direct

Bytes : 2
Cycles : 2

Encoding :

a10	a9	a8	0	0	0	0	1

a7	a6	a5	a4	a3	a2	a1	a0

Operation : AJMP
(PC) ← (PC) + 2
(PC_{10-0}) ← page address

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

Function : Logical-AND for byte variables

Description : ANL performs the bitwise logical-AND operation between the variables indicated and stores the results in the destination variable. No flags are affected. The two operands allow six addressing mode combinations. When the destination is the Accumulator, the source can use register, direct, register-indirect, or immediate addressing ; when the destination is a direct address, the source can be the Accumulator or immediate data. *Note* : When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

Example : If the Accumulator holds 0C3H (11000011B) and register 0 holds 55H (01010101B) then the instruction,
ANL A, R0
will leave 41H (01000001B) in the Accumulator.
When the destination is a directly addressed byte, this instruction will clear combinations of bits in any RAM location or hardware register. The mask byte determining the pattern of bits to be cleared would either be a constant contained in the instruction or a value computed in the Accumulator at run-time. The instruction,
ANL P1, #01110011B
will clear bits 7, 3, and 2 of output port 1.

ANL A, Rn

Bytes : 1
Cycles : 1

Encoding :

0	1	0	1	1	r	r	r

Operation : ANL
(A) ← (A) ∧ (Rn)

ANL A, direct

Bytes : 2
Cycles : 1

Encoding :

0	1	0	1	0	1	0	1

direct address

Operation : ANL
(A) ← (A) ∧ (direct)

ANL A, @ RI

Byte : 1

Cycle : 1

Encoding :

0	1	0	1	0	1	1	i

Opération : ANL
(A) ← (A) ∧ ((R_i))

ANL A, #DATA

Bytes : 2

Cycle : 1

Encoding :

0	1	0	1	0	1	0	0	immediate data

Operation : ANL
(A) ← (A) ∧ # data

ANL direct, A

Bytes : 2

Cycle : 1

Encoding :

0	1	0	1	0	0	1	0	direct address

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

ANL direct, # data

Bytes : 3

Cycles : 2

Encoding :

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

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

ANL C, <src-bit>

Function : Logical-AND for bit variables

Description : If the Boolean value of the source bit is 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 flags are affected.

Only direct addressing is allowed for the source operand.

Example : Set the carry flag 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 : ANL
$(C) \leftarrow (C) \wedge \overline{(bit)}$

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

Function : Compare and Jump if Not Equal

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

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

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

```
             CJNE    R7, #60H, NOT_EQ
;            ...     ...                 ; R7 = 60H
NOT_EQ :     JC      REQ_LOW             ; IF R7 < 60H
;            ...     ...                 ; R7 > 60H
```

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

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

```
WAIT : CJNE A, P1, WAIT
```

clears the carry flag and continues with the next instruction in sequence, since the Accumulator does equal the data read from P1. (If some other value was being input on 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) ← (PC) + 3
IF (A) <> (direct)
THEN
        (PC) ← (PC) + relative offset
IF (A) < (direct)
THEN
        (C) ← 1
ELSE
        (C) ← 0
```

CJNE A, # data, rel

Bytes : 3

Cycles : 2

Encoding :

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

Operation :

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

CJNE Rn, # data, rel

Bytes : 3

Cycles : 2

Encoding :

1	0	1	1	1	r	r	r	immediate data	rel. address

Operation :

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

CJNE @Ri, # data, rel

Bytes : 3

Cycles : 2

Encoding :

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

Operation :

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

CLR A

Function : Clear Accumulator

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

Example : The Accumulator contains 5CH (01011100B). The instruction,
CLRA
Will leave the Accumulator set to 00H (00000000B).

Bytes : 1

Cycles : 1

Encoding :

1	1	1	1	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

CPLA

Function : Complement Accumulator

Descritpion : 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,
CPLA
will leave the Accumulator set to 0A3H (10100011B).

Bytes : 1

Cycles : 1

Encoding :

1	1	1	1	0	1	0	0

Operation : CPL
$(A) \leftarrow \overline{(A)}$

CPL bit

Function : Complement bit

Description : The bit variable specified is complemented. A bit which had been a one is changed to zero and vice-versa. No other flags are affected. CLR can operate on the carry or any directly addressable bit.
Note : When this instruction is used to modify an output pin, the value used as the original data will be read from the output data latch, not the input pin.

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

CPL P1.1

CPL P1.2

will leave the port set to 5BH (01011011B).

CPL C

Bytes : 1

Cycles : 1

Encoding :

1	0	1	1	0	0	1	1

Operation : CPL
(C) $\leftarrow \overline{(C)}$

CPL bit

Bytes : 2

Cycles : 1

Encoding :

1	0	1	1	0	0	1	0	bit address

Operation : CPL
(bit) $\leftarrow \overline{(bit)}$

DA A

Function : Decimal-adjust Accumulator for Addition

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

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

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

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.

```
ADDCA, R3
DA          A
```

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

The decimal Adjust instruction will then after 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 set by the Decimal Adjust instruction, indicating that a decimal overflow occured. The true sum 56,67, and 1 is 124. BCD variables can be incremented or decremented by adding 01H or 99H. If the Accumulator initially holds 30H (representing the digits of 30 decimal), then the instruction sequence,

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

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

Bytes : 1

Cycles : 1

Encoding :

1	1	0	1	0	1	0	0

Operation :

```
DA
– contents of Accumulator are BCD
IF [[$(A_{3-0}) > 9$] V [(AC) = 1]]
        THEN $(A_{3-0}) \leftarrow (A_{3-0}) + 6$
                AND
IF [[$(A_{7-4}) > 9$] V [(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 7 EH and 7FH contain 00H and 40H, respectively. The instruction sequence.

DEC @ R0

DEC R0

DEC @ R0

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

DEC A

Bytes : 1

Cycles : 1

Encoding :

0	0	0	1	0	1	0	0

Operation : DEC
(A) ← (A) - 1

DEC Rn

Bytes : 1

Cycles : 1

Encoding :

0	0	0	1	1	r	r	r

Operation : DEC
(Rn) ← (Rn) - 1

DEC direct

Bytes : 2

Cycles : 1

Encoding :

0	0	0	1	0	1	0	1	direct address

Operation : DEC
(direct) ← (direct) - 1

DEC @ RI

Bytes : 1

Cycles : 1

Encoding :

0	0	0	1	0	1	1	i

Operation : DEC
((Ri)) ← ((Ri)) - 1

DIV AB

Function : Divide

Description : DIV AB divides the unsigned eight-bit integer in the Accumulator by the unsigned eight-bit integer in register B. The Accumulator receives the integer part of the quotient ; register B receives the integer remainder. The carry and OV flags will be cleared. *Exception* : If B had originally contained 00H ; the values returned in the Accumulator and B-register will be undefined and the overflow flag will be set. The carry flag is cleared in any case.

Example : The Accumulator contains 251 (0FBH 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 00010001B) in B, since 251 = (13 × 18) + 17. Carry and OV will both be cleared.

Bytes : 1

Cycles : 4

Encoding :

1	0	0	0	0	1	0	0

Operation : DIV

$(A)_{15-8}$

$\leftarrow (A)/(B)$

$(B)_{7-0}$

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 LABEL2 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 after the pin.

DJNZ Rn, rel

Bytes : 2

Cycles : 2

Encoding :

1	1	0	1	1	r	r	r	rel. address

Operation :

```
DJNZ
(PC) ← (PC) + 2
(Rn) ← (Rn) - 1
IF (RN) > 0 or (Rn) < 0
        THEN
                (PC) ← (PC) + rel
```

DJNZ direct, rel

Bytes : 3

Cycles : 2

Encoding :

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

Operation :

```
DJNZ
(PC) ← (PC) + 2
(direct) ← (direct) - 1
IF (direct) > 0 or (direct) < 0
        THEN
                (PC) ← (PC) + rel
```

INC <byte>

Function : Increment

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

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

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

INC @R0

INC R0

INC @R0

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

INC A

Bytes : 1

Cycles : 1

Encoding :

0	0	0	0	0	1	0	0

Operation :

```
INC
(A) ← (A) + 1
```

INC Rn

Bytes : 1

Cycles : 1

Encoding :

0	0	0	0	1	r	r	r

Operation :

```
INC
(Rn) ← (Rn) + 1
```

INC direct

Bytes : 2

Cycles : 1

Encoding :

0	0	0	0	0	1	0	1	direct address

Operation : INC
(direct) $\leftarrow$ (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 : Registers DPH and DPL contain 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

Descritpion : 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.

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

JB P1.2, LABEL 1 JB ACC.2, LABEL 2

will cause program execution to branch to the instruction at label LABEL 2.

Bytes : 3
Cycles : 2

Encoding :

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

Operation :

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

JBC bit, rel

Function : Jump if Bit is set and Clear bit

Description : If the indicated bit is a 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, LABEL 1
JBC ACC.2, LABEL 2
```

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

Bytes : 3
Cycles : 2

Encoding :

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

Operation :

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

JC rel

Function : Jump if Carry is set

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

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

```
JC      LABEL 1
CPL     C
JC      LABEL 2
```

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

Bytes : 2
Cycles : 2

Encoding :

0	1	0	0	0	0	0	0	rel. address

Operation :

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

JMP @A + DPTR

Function : Jump indirect

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

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

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

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

Bytes : 1

Cycles : 2

Encoding :

0	1	1	1	0	0	1	1

Operation : JMP
(PC) ← (A) + (DPTR)

JNB bit, rel

Function : Jump if Bit 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 ACC3, LABEL2
```

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

Bytes : 3

Cycles : 2

Encoding :

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

Operation :

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

JNC rel

Function : Jump if Carry not set

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

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

```
JNC LABEL1
CPLC
JNC LABEL2
```

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

Bytes : 2

Cycles : 2

Encoding :

0	1	0	1	0	0	0	0		rel. address

Operation :

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

JNZ rel

Function : Jump if Accumulator Not Zero

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

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

```
JNZ LABEL1
INC A
JNZ LABEL2
```

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

Bytes : 2

Cycles : 2

Encoding :

0	1	1	1	0	0	0	0		rel. address

Operation :

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

JZ rel

Function : Jump if Accumulator Zero

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

Example : The Accumulator originally contains 01H. The instruction sequence.

```
JZ      LABEL1
DEC     A
JZ      LABEL2
```

will change the Accumulator to 00H and cause program execution 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 $(PC) \leftarrow (PC) + rel$
```

LCALL addr16

Function : Long call

Description : LCALL calls a subroutine located 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 1235H.

Bytes : 3

Cycles : 2

Encoding :

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

Operation :

```
LCALL
$(PC) \leftarrow (PC) + 3$
$(SP) \leftarrow (SP) + 1$
$((SP)) \leftarrow (PC_{7-0})$
$(SP) \leftarrow (SP) + 1$
$((SP)) \leftarrow (SP_{15-8})$
$(PC) \leftarrow addr_{15-0}$
```

LJMP addr16

Function : Long Jump

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

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

LJMP JMPADR

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

Bytes : 3

Cycles : 2

Encoding :

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

Operation : LJMP
(PC) ← $addr_{15\text{-}0}$

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

Function : Move byte variable

Description : The byte variable indicated 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 combinaisons 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     R, @ R1          ; B <= 10h
MOV     @ R1, P1         ; RAM (40H) <= OCAH
MOV     P2, P1           ; P2 # 0CAH
```

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

MOV A, Rn

Bytes : 1

Cycles : 1

Encoding :

1	1	1	0	1	r	r	r

Operation : MOV
(A) ← (Rn)

***MOV A,direct**

Bytes : 2

Cycles : 1

Encoding :

1	1	1	0	0	1	0	1	direct address

Operation : MOV
(A) ← (direct)

*MOV A, ACC is not valid instruction.

MOV A,@ RI

Bytes : 1

Cycles : 1

Encoding :

1	1	1	0	0	1	1	i

Operation : MOV
(A) ← (Ri)

MOV A, # data

Bytes : 2

Cycles : 1

Encoding :

0	1	1	1	0	1	0	0	immediate data

Operation : MOV
(A) ← # data

MOV Rn, A

Bytes : 1

Cycles : 1

Encoding :

1	1	1	1	1	r	r	r

Operation : MOV
(Rn) ← (A)

MOV Rn, direct

Bytes : 2

Cycles : 2

Encoding :

1	0	1	0	1	r	r	r	direct addr.

Operation : MOV
(Rn) ← (direct)

MOV Rn, # data

Bytes : 2

Cycles : 1

Encoding :

0	1	1	1	1	r	r	r	immediate data

Operation : MOV
(Rn) ← # data

MOV direct, A

Bytes : 2

Cycles : 1

Encoding :

1	1	1	1	0	1	0	1	direct address

Operation : MOV
(direct) ← (A)

MOV direct, Rn

Bytes : 2

Cycles : 2

Encoding :

1	0	0	0	1	r	r	r	direct address

Operation : MOV
(direct) ← (Rn)

MOV direct, direct

Bytes : 3

Cycles : 2

Encoding :

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

Operation : MOV
(direct) ← (direct)

MOV direct, @ Ri

Bytes : 2

Cycles : 2

Encoding :

1	0	0	0	0	1	1	i	direct addr.

Operation : MOV
(direct) ← (Ri)

MOV direct, # data

Bytes : 3

Cycles : 2

Encoding :

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

Operation : MOV
(direct) ← # data

MOV @ Ri, A

Bytes : 1

Cycles : 1

Encoding :

1	1	1	1	0	1	1	i

Operation : MOV
((Ri)) ← (A)

MOV @ Ri, direct

Bytes : 2

Cycles : 2

Encoding :

1	0	1	0	0	1	1	i	direct addr.

Operation : MOV
((Ri)) ← (direct)

MOV @ Ri*, data

Bytes : 2

Cycles : 1

Encoding :

0	1	1	1	0	1	1	i	immediate data

Operation : MOV
((Ri)) ← # data

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

Function : More 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	0	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	immed. data 15-8	immed. data 7-0

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

MOVC A, @ A + <base-reg>

Function : Move Code byte

Description : The MOVC instructions load the Accumulator with a code byte, or constant from program memory. The address of the byte fetched is the sum of the original unsigned eight-bit. Accumulator contents and the contents of a sixteen-bit base register, which may be either the Data Pointer or the PC. In the latter case, 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
           CB         77H
           CB         88H
           DB         99H
```

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

MOVC A, @ A + DPTR

Bytes : 1

Cycles : 2

Encoding :

1	0	0	1	0	0	1	1

Operation : MOVC
(A) ← ((A) + (DPTR))

MOVC A, @ A + PC

Bytes : 1

Cycles : 2

Encoding :

1	0	0	0	0	0	1	1

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

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

Function : Move External

Description : The MOVX instructions transfer data between the Accumulator and a byte of external data memory, hence the "X" appended to MOV. There are two types of instructions, differing in whether they provide an eight-bit or sixteen-bit indirect address to the external data RAM.
In the first type, the contents of R0 or R1 in the current register bank provide an eight-bit address multiplexed with data on P0. Eight bits are sufficient for external I/O expansion decoding or for a relatively small RAM array. For somewhat larger arrays, any output port pins can be used to output higher-order address bits. These pins would be controlled by an output instruction preceding the MOVX.
In the second type of MOVX instruction, the Data Pointer generates a sixteen-bit address. P2 outputs the high-order eight address bits (the contents of DPH) while P0 multiplexes the low-order eight bits (DPL) with data. The P2 Special Function Register retains its previous contents while the P2 output buffers are emitting the contents of DPH. This form is faster and more efficient when accessing very large data arrays (up to 64K bytes), since no additional instructions are needed to set up the output ports.
It is possible in some situation 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 is connected to the 80C51 Port 0. Port 3 provides control lines for the external RAM. Ports 0 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 @ Ri, A

Bytes : 1
Cycles : 2

Encoding :

1	1	1	1	0	0	1	i

Operation : MOVX
$((Ri)) \leftarrow (A)$

MOVX A, @ DPTR

Bytes : 1
Cycles : 2

Encoding :

1	1	1	0	0	0	0	0

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

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 (OFFH) 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 (OAOH). 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 continue at the following instruction. Other than the PC, no registers or flags are effected.
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 enable) with the instruction sequence.

```
CLR      P2.7
NOP
NOP
NOP
NOP
SETP     P2.7
```

Bytes :	1
Cycles :	1
Encoding :	0 0 0 0 | 0 0 0 0
Operation :	NOP $(PC) \leftarrow (PC) + 1$

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 combinaisons. 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) ← (A) V (Rn)

ORL A, direct

Bytes : 2
Cycles : 1

Encoding :

0	1	0	0	0	1	0	1	direct address

Operation : ORL
(A) ← (A) V (direct)

ORL A, @ Ri

Bytes : 1
Cycles : 1

Encoding :

0	1	0	0	0	1	1	i

Operation : ORL
(A) ← (A) V ((Ri))

ORL A, # data

Bytes : 2
Cycles : 1

Encoding :

0	1	0	0	0	1	0	0	immediate data

Operation : ORL
(A) ← (A) V # data

ORL direct, A

Bytes : 2
Cycles : 1

Encoding :

0	1	0	0	0	0	1	0	direct address

Operation : ORL
(direct) ← (direct) V (A)

ORL direct, # data

Bytes : 3

Cycles : 2

Encoding :

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

Operation : ORL
(direct) ← (direct) V # data

ORL C, <src-bit>

Function : Logical-OR for bit variable

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 it self 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. BIT7
ORL     C,/OV         ; OR CARRY WITH THE INVERSE OF OV
```

ORL C, bit

Bytes : 2

Cycles : 2

Encoding :

0	1	1	1	0	0	1	0	bit address

Operation : ORL
(C) ← (C) V (bit)

ORL C, /bit

Bytes : 2

Cycles : 2

Encoding :

1	0	1	0	0	0	0	0	bit address

Operation : ORL
(C) ← (C) V $\overline{\text{(bit)}}$

POP direct

Function : Pop from stack.

Description : The contents of 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
(direct) ← ((SP))
(SP) ← (SP) – 1

PUSH direct

Function : push onto stack.

Description : The Stack Pointer is incremented by one. The contents fo 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 location 0AH and 0BH, respectively.

Bytes : 2

Cycles : 2

Encoding :

1	1	0	0	0	0	0	0	direct address

Operation : PUSH
(SP) ← (SP) + 1
((SP)) ← (direct)

RET

Function : Return from subroutine

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

Example : The Stack Pointer originally contains the value 0BH. Internal RAM locations 0AH and 0BH contain the values 23H, and 01H, respectively. The instruction,
RET
will leave the Stack Pointer equal to the value 09H. Program execution will continue at location 0123H.

Bytes : 1

Cycles : 2

Encoding :

0	0	1	0	0	0	1	0

Operation : RET
(PC_{15-8}) ← ((SP))
(SP) ← (SP) – 1
(PC7-0) ← ((SP))
(SP) ← (SP) – 1

RETI

Function : Return from interrupt

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

Example : The Stack Pointer originally contains the value 0BH. An interrupt was detected during the instruction ending at location 0122H. Internal RAM locations 0AH and 0BH contain the values 23H and 01H, respectively. The instruction,
RETI
will leave the Stack Pointer equal to 09H and return program execution to location 0123H.

Bytes :	1
Cycles :	2
Encoding :	0 0 1 1 | 0 0 1 0
Operation :	RETI $(PC_{15-8}) \leftarrow ((SP))$ $(SP) \leftarrow (SP) - 1$ $(PC_{7-0}) \leftarrow ((SP))$ $(SP) \leftarrow (SP) - 1$

RL A

Function :	Rotate Accumulator Left
Description :	The eight bits in the Accumulator are rotated one bit to the left. Bit 7 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 (100001011B) with the carry unaffected.
Bytes :	1
Cycles :	1
Encoding :	0 0 1 0 | 0 0 1 1
Operation :	RL $(An + 1) \leftarrow (An)$ $n = 0 - 6$ $(A0) \leftarrow (A7)$

RLC A

Function :	Rotate Accumulator Left through the Carry flag
Description :	The eight bits in the Accumulator and the carry flag are together rotated one bit to the left. Bit 7 moves into the carry flag ; the original state of the carry flag moves into the bit 0 position. No other flags are affected.
Example :	The Accumulator holds the value 0C5H (11000101B), and the carry is zero. The instruction, RCL A leaves the Accumulator holding the value 8BH (10001010B) with the carry set.
Bytes :	1
Cycles :	1
Encoding :	0 0 1 1 | 0 0 1 1
Operation :	RLC $(An + 1) \leftarrow (An)$ $n = 0 - 6$ $(A0) \leftarrow (C)$ $(C) \leftarrow (A7)$

RR A

Function : Rotate Accumulator Right

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

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

RR A

leaves the Accumulator holding the value 0E2H (11100010B) with the carry unaffected.

Bytes : 1

Cycles : 1

Encoding :

0	0	0	0	0	0	1	1

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

RRC A

Function : Rotate Accumulator Right through 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 62 (01100010B) with the carry set.

Bytes : 1

Cycles : 1

Encoding :

0	0	0	1	0	0	1	1

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

SETB <bit>

Function : Set bit

Description : SETB sets the indicated bit to one. SETB can operate on the carry flag or any direct 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 128 bytes 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

SETB <bit>

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 substraction 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 substraction, it should not be explicity cleared by a CLRC instruction.

SUBB A, Rn

Bytes : 1

Cycles : 1

Encoding :

1	0	0	1	1	r	r	r

Operation : SUBB
(A) ← (A) – (C) – (Rn)

SUBB A, direct

Bytes : 2

Cycles : 1

Encoding :

1	0	0	1	0	1	0	1	direct address

Operation : SUBB
(A) ← (A) – (C) – (direct)

SUBB A, @ Ri

Bytes : 1

Cycles : 1

Encoding :

1	0	0	1	0	1	1	i

Operation : SUBB
(A) ← (A) – (C) – (Ri)

SUBB A, # data

Bytes : 2

Cycles : 1

Encoding :

1	0	0	1	0	1	0	0	immediate data

Operation : SUBB
(A) ← (A) – (C) – # data

SWAP A

Function : Swap nibbles within the Accumulator

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

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

SWAP A

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

Bytes : 1

Cycles : 1

Encoding :

1	1	0	0	0	1	0	0

Operation : SWAP

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

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 addres 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

$(A) \leftrightarrows (Rn)$

XCH A, direct

Bytes : 2

Cycles : 1

Encoding :

1	1	0	0	0	1	0	1	direct address

Operation : XCH

$(A) \leftrightarrows (direct)$

XCH A, @Ri

Bytes : 1

Cycles : 1