MCS®-51 INSTRUCTION SET

Table 10. 8051 Instruction Set Summary

Interrupt Response Time: Refer to Hardware Description Chapter.

Instructions that Affect Flag Settings[1]

Instruction	Flag			Instruction	Flag		
	C	OV	AC		C	OV	AC
ADD	X	X	X	CLR C	O		
ADDC	X	X	X	CPL C	X		
SUBB	X	X	X	ANL C,bit	X		
MUL	O	X		ANL C,/bit	X		
DIV	O	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) addressed 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. A branch 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 conditional 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	Oscillator 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
ADDC	A,Rn	Add register to Accumulator with Carry	1	12
ADDC	A,direct	Add direct byte to Accumulator with Carry	2	12
ADDC	A,@Ri	Add indirect RAM to Accumulator with Carry	1	12
ADDC	A,#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

All mnemonics copyrighted © Intel Corporation 1980

Table 10. 8051 Instruction Set Summary (Continued)

Mnemonic		Description	Byte	Oscillator Period
ARITHMETIC OPERATIONS (Continued)				
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
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

Mnemonic		Description	Byte	Oscillator Period
LOGICAL OPERATIONS (Continued)				
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
DATA TRANSFER				
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

All mnemonics copyrighted © Intel Corporation 1980

Table 10. 8051 Instruction Set Summary (Continued)

Mnemonic		Description	Byte	Oscillator Period
DATA TRANSFER (Continued)				
MOV	@Ri,direct	Move direct byte 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
MOVC	A,@A+DPTR	Move Code byte relative to DPTR to Acc	1	24
MOVC	A,@A+PC	Move Code byte relative to PC to Acc	1	24
MOVX	A,@Ri	Move External RAM (8-bit addr) to Acc	1	24
MOVX	A,@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 onto stack	2	24
POP	direct	Pop direct byte from stack	2	24
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 low-order Digit indirect RAM with Acc	1	12

Mnemonic		Description	Byte	Oscillator Period
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
PROGRAM BRANCHING				
ACALL	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
AJMP	addr11	Absolute Jump	2	24
LJMP	addr16	Long Jump	3	24
SJMP	rel	Short Jump (relative addr)	2	24

All mnemonics copyrighted © Intel Corporation 1980

Table 10. 8051 Instruction Set Summary (Continued)

Mnemonic		Description	Byte	Oscillator Period
PROGRAM BRANCHING (Continued)				
JMP	@A+DPTR	Jump indirect 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
CJNE	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

Mnemonic		Description	Byte	Oscillator Period
PROGRAM BRANCHING (Continued)				
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

All mnemonics copyrighted © Intel Corporation 1980

Table 11. Instruction Opcodes in Hexadecimal Order

Hex Code	Number of Bytes	Mnemonic	Operands		Hex Code	Number of Bytes	Mnemonic	Operands
00	1	NOP			33	1	RLC	A
01	2	AJMP	code addr		34	2	ADDC	A,#data
02	3	LJMP	code addr		35	2	ADDC	A,data addr
03	1	RR	A		36	1	ADDC	A,@R0
04	1	INC	A		37	1	ADDC	A,@R1
05	2	INC	data addr		38	1	ADDC	A,R0
06	1	INC	@R0		39	1	ADDC	A,R1
07	1	INC	@R1		3A	1	ADDC	A,R2
08	1	INC	R0		3B	1	ADDC	A,R3
09	1	INC	R1		3C	1	ADDC	A,R4
0A	1	INC	R2		3D	1	ADDC	A,R5
0B	1	INC	R3		3E	1	ADDC	A,R6
0C	1	INC	R4		3F	1	ADDC	A,R7
0D	1	INC	R5		40	2	JC	code addr
0E	1	INC	R6		41	2	AJMP	code addr
0F	1	INC	R7		42	2	ORL	data addr,A
10	3	JBC	bit addr, code addr		43	3	ORL	data addr,#data
11	2	ACALL	code addr		44	2	ORL	A,#data
12	3	LCALL	code addr		45	2	ORL	A,data addr
13	1	RRC	A		46	1	ORL	A,@R0
14	1	DEC	A		47	1	ORL	A,@R1
15	2	DEC	data addr		48	1	ORL	A,R0
16	1	DEC	@R0		49	1	ORL	A,R1
17	1	DEC	@R1		4A	1	ORL	A,R2
18	1	DEC	R0		4B	1	ORL	A,R3
19	1	DEC	R1		4C	1	ORL	A,R4
1A	1	DEC	R2		4D	1	ORL	A,R5
1B	1	DEC	R3		4E	1	ORL	A,R6
1C	1	DEC	R4		4F	1	ORL	A,R7
1D	1	DEC	R5		50	2	JNC	code addr
1E	1	DEC	R6		51	2	ACALL	code addr
1F	1	DEC	R7		52	2	ANL	data addr,A
20	3	JB	bit addr, code addr		53	3	ANL	data addr,#data
21	2	AJMP	code addr		54	2	ANL	A,#data
22	1	RET			55	2	ANL	A,data addr
23	1	RL	A		56	1	ANL	A,@R0
24	2	ADD	A,#data		57	1	ANL	A,@R1
25	2	ADD	A,data addr		58	1	ANL	A,R0
26	1	ADD	A,@R0		59	1	ANL	A,R1
27	1	ADD	A,@R1		5A	1	ANL	A,R2
28	1	ADD	A,R0		5B	1	ANL	A,R3
29	1	ADD	A,R1		5C	1	ANL	A,R4
2A	1	ADD	A,R2		5D	1	ANL	A,R5
2B	1	ADD	A,R3		5E	1	ANL	A,R6
2C	1	ADD	A,R4		5F	1	ANL	A,R7
2D	1	ADD	A,R5		60	2	JZ	code addr
2E	1	ADD	A,R6		61	2	AJMP	code addr
2F	1	ADD	A,R7		62	2	XRL	data addr,A
30	3	JNB	bit addr, code addr		63	3	XRL	data addr,#data
31	2	ACALL	code addr		64	2	XRL	A,#data
32	1	RETI			65	2	XRL	A,data addr

Table 11. Instruction Opcodes in Hexadecimal Order (Continued)

Hex Code	Number of Bytes	Mnemonic	Operands	Hex Code	Number of Bytes	Mnemonic	Operands
66	1	XRL	A,@R0	99	1	SUBB	A,R1
67	1	XRL	A,@R1	9A	1	SUBB	A,R2
68	1	XRL	A,R0	9B	1	SUBB	A,R3
69	1	XRL	A,R1	9C	1	SUBB	A,R4
6A	1	XRL	A,R2	9D	1	SUBB	A,R5
6B	1	XRL	A,R3	9E	1	SUBB	A,R6
6C	1	XRL	A,R4	9F	1	SUBB	A,R7
6D	1	XRL	A,R5	A0	2	ORL	C,/bit addr
6E	1	XRL	A,R6	A1	2	AJMP	code addr
6F	1	XRL	A,R7	A2	2	MOV	C,bit addr
70	2	JNZ	code addr	A3	1	INC	DPTR
71	2	ACALL	code addr	A4	1	MUL	AB
72	2	ORL	C,bit addr	A5		reserved	
73	1	JMP	@A+DPTR	A6	2	MOV	@R0,data addr
74	2	MOV	A,#data	A7	2	MOV	@R1,data addr
75	3	MOV	data addr,#data	A8	2	MOV	R0,data addr
76	2	MOV	@R0,#data	A9	2	MOV	R1,data addr
77	2	MOV	@R1,#data	AA	2	MOV	R2,data addr
78	2	MOV	R0,#data	AB	2	MOV	R3,data addr
79	2	MOV	R1,#data	AC	2	MOV	R4,data addr
7A	2	MOV	R2,#data	AD	2	MOV	R5,data addr
7B	2	MOV	R3,#data	AE	2	MOV	R6,data addr
7C	2	MOV	R4,#data	AF	2	MOV	R7,data addr
7D	2	MOV	R5,#data	B0	2	ANL	C,/bit addr
7E	2	MOV	R6,#data	B1	2	ACALL	code addr
7F	2	MOV	R7,#data	B2	2	CPL	bit addr
80	2	SJMP	code addr	B3	1	CPL	C
81	2	AJMP	code addr	B4	3	CJNE	A,#data,code addr
82	2	ANL	C,bit addr	B5	3	CJNE	A,data addr,code addr
83	1	MOVC	A,@A+PC	B6	3	CJNE	@R0,#data,code addr
84	1	DIV	AB	B7	3	CJNE	@R1,#data,code addr
85	3	MOV	data addr, data addr	B8	3	CJNE	R0,#data,code addr
86	2	MOV	data addr,@R0	B9	3	CJNE	R1,#data,code addr
87	2	MOV	data addr,@R1	BA	3	CJNE	R2,#data,code addr
88	2	MOV	data addr,R0	BB	3	CJNE	R3,#data,code addr
89	2	MOV	data addr,R1	BC	3	CJNE	R4,#data,code addr
8A	2	MOV	data addr,R2	BD	3	CJNE	R5,#data,code addr
8B	2	MOV	data addr,R3	BE	3	CJNE	R6,#data,code addr
8C	2	MOV	data addr,R4	BF	3	CJNE	R7,#data,code addr
8D	2	MOV	data addr,R5	C0	2	PUSH	data addr
8E	2	MOV	data addr,R6	C1	2	AJMP	code addr
8F	2	MOV	data addr,R7	C2	2	CLR	bit addr
90	3	MOV	DPTR,#data	C3	1	CLR	C
91	2	ACALL	code addr	C4	1	SWAP	A
92	2	MOV	bit addr,C	C5	2	XCH	A,data addr
93	1	MOVC	A,@A+DPTR	C6	1	XCH	A,@R0
94	2	SUBB	A,#data	C7	1	XCH	A,@R1
95	2	SUBB	A,data addr	C8	1	XCH	A,R0
96	1	SUBB	A,@R0	C9	1	XCH	A,R1
97	1	SUBB	A,@R1	CA	1	XCH	A,R2
98	1	SUBB	A,R0	CB	1	XCH	A,R3

Table 11. Instruction Opcodes in Hexadecimal Order (Continued)

Hex Code	Number of Bytes	Mnemonic	Operands		Hex Code	Number of Bytes	Mnemonic	Operands
CC	1	XCH	A,R4		E6	1	MOV	A,@R0
CD	1	XCH	A,R5		E7	1	MOV	A,@R1
CE	1	XCH	A,R6		E8	1	MOV	A,R0
CF	1	XCH	A,R7		E9	1	MOV	A,R1
D0	2	POP	data addr		EA	1	MOV	A,R2
D1	2	ACALL	code addr		EB	1	MOV	A,R3
D2	2	SETB	bit addr		EC	1	MOV	A,R4
D3	1	SETB	C		ED	1	MOV	A,R5
D4	1	DA	A		EE	1	MOV	A,R6
D5	3	DJNZ	data addr,code addr		EF	1	MOV	A,R7
D6	1	XCHD	A,@R0		F0	1	MOVX	@DPTR,A
D7	1	XCHD	A,@R1		F1	2	ACALL	code addr
D8	2	DJNZ	R0,code addr		F2	1	MOVX	@R0,A
D9	2	DJNZ	R1,code addr		F3	1	MOVX	@R1,A
DA	2	DJNZ	R2,code addr		F4	1	CPL	A
DB	2	DJNZ	R3,code addr		F5	2	MOV	data addr,A
DC	2	DJNZ	R4,code addr		F6	1	MOV	@R0,A
DD	2	DJNZ	R5,code addr		F7	1	MOV	@R1,A
DE	2	DJNZ	R6,code addr		F8	1	MOV	R0,A
DF	2	DJNZ	R7,code addr		F9	1	MOV	R1,A
E0	1	MOVX	A,@DPTR		FA	1	MOV	R2,A
E1	2	AJMP	code addr		FB	1	MOV	R3,A
E2	1	MOVX	A,@R0		FC	1	MOV	R4,A
E3	1	MOVX	A,@R1		FD	1	MOV	R5,A
E4	1	CLR	A		FE	1	MOV	R6,A
E5	2	MOV	A,data addr		FF	1	MOV	R7,A

INSTRUCTION DEFINITIONS

ACALL addr11

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

ADD A,<src-byte>

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

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

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

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

ADD A,R0

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

ADD A,Rn

Bytes:	1
Cycles:	1
Encoding:	`0 0 1 0` `1 r r r`
Operation:	ADD
	(A) ← (A) + (Rn)

ADD A,direct

Bytes:	2
Cycles:	1
Encoding:	`0 0 1 0` `0 1 0 1` `direct address`
Operation:	ADD
	(A) ← (A) + (direct)

ADD   A,@Ri

        Bytes: 1

      Cycles: 1

   Encoding:

0 0 1 0	0 1 1 i

  Operation: ADD
$(A) \leftarrow (A) + ((R_i))$

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: 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 flags are set, respectively, if there is a carry-out from bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occured.

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

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

       Example: The Accumulator holds 0C3H (11000001B) 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

    Bytes: 1

    Cycles: 1

    Encoding: | 0 0 1 1 | 1 r r r |

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

ADDC   A,direct

    Bytes: 2

    Cycles: 1

    Encoding: | 0 0 1 1 | 0 1 0 1 | | direct address |

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

ADDC   A,@Ri

    Bytes: 1

    Cycles: 1

    Encoding: | 0 0 1 1 | 0 1 1 i |

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

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 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 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
$(PC) \leftarrow (PC) + 2$
$(PC_{10-0}) \leftarrow$ 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

>**Bytes:**   1

>**Cycles:**   1

>**Encoding:**   `0 1 0 1 | 0 1 1 i`

>**Operation:**   ANL
>
>(A) ← (A) ∧ ((Ri))

ANL   A,#data

>**Bytes:**   2

>**Cycles:**   1

>**Encoding:**   `0 1 0 1 | 0 1 0 0`   `immediate data`

>**Operation:**   ANL
>(A) ← (A) ∧ #data

ANL   direct,A

>**Bytes:**   2

>**Cycles:**   1

>**Encoding:**   `0 1 0 1 | 0 0 1 0`   `direct address`

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

ANL   direct,#data

Bytes:	3
Cycles:	2

Encoding: | 0 1 0 1 | 0 0 1 1 | | direct address | | immediate data |

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

ANL   C, <src-bit>

Function:	Logical-AND for bit variables
Description:	If the Boolean value of the source bit is a logical 0 then clear the carry flag; otherwise leave the carry flag in its current state. A slash ("/") preceding the operand in the assembly language indicates that the logical complement of the addressed bit is used as the source value, *but the source bit itself is not affected.* No other flags are affected.
	Only direct addressing is allowed for the source operand.
Example:	Set the carry flag if, and only if, P1.0 = 1, ACC. 7 = 1, and OV = 0:

      MOV   C,P1.0   ;LOAD CARRY WITH INPUT PIN STATE

      ANL   C,ACC.7   ;AND CARRY WITH ACCUM. BIT 7

      ANL   C,/OV   ;AND WITH INVERSE OF OVERFLOW FLAG

ANL   C,bit

Bytes:	2
Cycles:	2

Encoding: | 1 0 0 0 | 0 0 1 0 | | bit address |

Operation: ANL
(C) ← (C) ∧ (bit)

ANL   C,/bit

Bytes:	2
Cycles:	2

Encoding: | 1 0 1 1 | 0 0 0 0 | | bit address |

Operation: ANL
(C) ← (C) ∧ ⌐   (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,

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 (10100011B).

Bytes:	1
Cycles:	1

Encoding:

1 1 1 1	0 1 0 0

Operation: CPL
$(A) \leftarrow \neg (A)$

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
$(C) \leftarrow \neg (C)$

CPL bit

Bytes: 2

Cycles: 1

Encoding:

| 1 0 1 1 | 0 0 1 0 | | bit address |

Operation: CPL
(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 $[[(A_{3\text{-}0}) > 9] \lor [(AC) = 1]]$
                     THEN$(A_{3\text{-}0}) \leftarrow (A_{3\text{-}0}) + 6$
                           AND

              IF $[[(A_{7\text{-}4}) > 9] \lor [(C) = 1]]$
                     THEN $(A_{7\text{-}4}) \leftarrow (A_{7\text{-}4}) + 6$

DEC byte

Function:	Decrement
Description:	The variable indicated is decremented by 1. An original value of 00H will underflow to 0FFH. No flags are affected. Four operand addressing modes are allowed: accumulator, register, direct, or register-indirect.

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

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

DEC @R0

DEC R0

DEC @R0

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

DEC A

Bytes:	1
Cycles:	1

Encoding:

0 0 0 1	0 1 0 0

Operation: DEC
(A) ← (A) − 1

DEC Rn

Bytes:	1
Cycles:	1

Encoding:

0 0 0 1	1 r r r

Operation: DEC
(Rn) ← (Rn) − 1

DEC   direct

Bytes:	2
Cycles:	1

Encoding: | 0 0 0 1 | 0 1 0 1 | | direct address |

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

DEC   @Ri

Bytes:	1
Cycles:	1

Encoding: | 0 0 0 1 | 0 1 1 i |

Operation: DEC
((Ri)) ← ((Ri)) − 1

DIV   AB

Function: Divide

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

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

Example: The Accumulator contains 251 (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 X 18) + 17. Carry and OV will both be cleared.

Bytes:	1
Cycles:	4

Encoding: | 1 0 0 0 | 0 1 0 0 |

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

DJNZ   <byte>,<rel-addr>

Function:	Decrement and Jump if Not Zero
Description:	DJNZ decrements the location indicated by 1, and branches to the address indicated by the second operand if the resulting value is not zero. An original value of 00H will underflow to 0FFH. No flags are 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) ← (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
$(PC) \leftarrow (PC) + 2$
$(direct) \leftarrow (direct) - 1$
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) ← (Rn) + 1

INC   direct

Bytes:	2
Cycles:	1

Encoding:

0 0 0 0	0 1 0 1		direct address

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

INC   @Ri

Bytes:	1
Cycles:	1

Encoding:

0 0 0 0	0 1 1 i

Operation: INC
((Ri)) ← ((Ri)) + 1

INC   DPTR

Function: Increment Data Pointer

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

This is the only 16-bit register which can be incremented.

Example: 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) ← (DPTR) + 1

JB   bit,rel

Function:	Jump if Bit set
Description:	If the indicated bit is a one, jump to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. *The bit tested is not modified.* No flags are affected.
Example:	The data present at input port 1 is 11001010B. The Accumulator holds 56 (01010110B). The instruction sequence,

JB   P1.2,LABEL1

JB   ACC.2,LABEL2

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

Bytes:	3
Cycles:	2
Encoding:	`0 0 1 0` `0 0 0 0` `bit address` `rel. address`

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

JBC   bit,rel

Function:	Jump if Bit is set and Clear bit
Description:	If the indicated bit is one, branch to the address indicated; otherwise proceed with the next instruction. *The bit 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
            $(PC) \leftarrow (PC) + rel$

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     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
$(PC) \leftarrow (PC) + 2$
IF   (C) = 1
      THEN
            $(PC) \leftarrow (PC) + rel$

JMP @A+DPTR

Function:	Jump indirect
Description:	Add the eight-bit unsigned contents of the Accumulator with the sixteen-bit data pointer, and load the resulting sum to the program counter. This will be the address for subsequent instruction fetches. Sixteen-bit addition is performed (modulo 2^{16}): a carry-out from the low-order eight bits propagates through the higher-order bits. Neither the Accumulator nor the Data Pointer is altered. No flags are affected.
Example:	An even number from 0 to 6 is in the Accumulator. The following sequence of instructions will branch to one of four AJMP instructions in a jump table starting at JMP__TBL:

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

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

Bytes:	1
Cycles:	2
Encoding:	0 1 1 1 | 0 0 1 1
Operation:	JMP (PC) ← (A) + (DPTR)

JNB   bit,rel

Function:	Jump if Bit Not set
Description:	If the indicated bit is a zero, branch to the indicated address; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. *The bit tested is not modified.* No flags are affected.
Example:	The data present at input port 1 is 11001010B. The Accumulator holds 56H (01010110B). The instruction sequence,

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

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

Bytes:	3
Cycles:	2
Encoding:	`0 0 1 1` `0 0 0 0` `bit address` `rel. address`
Operation:	JNB (PC) ← (PC) + 3 IF (bit) = 0 THEN (PC) ← (PC) + rel.

JNC   rel

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

        JNC   LABEL1
        CPL   C
        JNC   LABEL2

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

Bytes:	2
Cycles:	2
Encoding:	`0 1 0 1` `0 0 0 0` `rel. address`
Operation:	JNC (PC) ← (PC) + 2 IF (C) = 0 THEN (PC) ← (PC) + rel

JNZ rel

Function:	Jump if Accumulator Not Zero
Description:	If any bit of the Accumulator is a one, branch to the indicated address; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. The Accumulator is not modified. No flags are affected.
Example:	The Accumulator originally holds 00H. The instruction sequence,

```
JNZ   LABEL1
INC   A
JNZ   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

$(PC) \leftarrow (PC) + 2$
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    LABEL2
```

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 $(PC) \leftarrow (PC) + rel$

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 1234H.

Bytes:	3
Cycles:	2

Encoding:

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

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

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
$(PC) \leftarrow addr_{15-0}$

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

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

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

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

```
MOV   R0,#30H    ;R0 < = 30H
MOV   A,@R0      ;A < = 40H
MOV   R1,A       ;R1 < = 40H
MOV   B,@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) ← (Rn)

*MOV   A,direct

Bytes:	2
Cycles:	1

Encoding:

1 1 1 0	0 1 0 1		direct address

Operation: MOV
(A) ← (direct)

MOV A,ACC is not a valid instruction.

MOV   A,@Ri

       Bytes: 1 .

       Cycles: 1

   Encoding: | 1 1 1 0 | 0 1 1 i |

   Operation: MOV
                  (A) ← ((Ri))

MOV   A,#data

       Bytes: 2

       Cycles: 1

   Encoding: | 0 1 1 1 | 0 1 0 0 | | immediate data |

   Operation: MOV
                  (A) ← #data

MOV   Rn,A

       Bytes: 1

       Cycles: 1

   Encoding: | 1 1 1 1 | 1 r r r |

   Operation: MOV
                  (Rn) ← (A)

MOV   Rn,direct

       Bytes: 2

       Cycles: 2

   Encoding: | 1 0 1 0 | 1 r r r | | direct addr. |

   Operation: MOV
                  (Rn) ← (direct)

MOV   Rn,#data

       Bytes: 2

       Cycles: 1

   Encoding: | 0 1 1 1 | 1 r r r | | immediate data |

   Operation: MOV
                  (Rn) ← #data

MOV   direct,A

    Bytes: 2

    Cycles: 1

    Encoding: | 1 1 1 1 | 0 1 0 1 | | direct address |

    Operation: MOV
                   (direct) ← (A)

MOV   direct,Rn

    Bytes: 2

    Cycles: 2

    Encoding: | 1 0 0 0 | 1 r r r | | direct address |

    Operation: MOV
                   (direct) ← (Rn)

MOV   direct,direct

    Bytes: 3

    Cycles: 2

    Encoding: | 1 0 0 0 | 0 1 0 1 | | dir. addr. (src) | | 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:**   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 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,#data16

    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. data15-8 | | immed. data7-0 |

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

MOVC A,@A+ <base-reg>

Function:	Move Code byte
Description:	The MOVC instructions load the Accumulator with a code byte, or constant from program memory. The address of the byte fetched is the sum of the original unsigned eight-bit Accumulator contents and the contents of a sixteen-bit base register, which may be either the Data Pointer or the PC. In the latter case, the PC is incremented to the address of the following instruction before being added with the Accumulator; otherwise the base register is not altered. Sixteen-bit addition is performed so a carry-out from the low-order eight bits may propagate through higher-order bits. No flags are affected.
Example:	A value between 0 and 3 is in the Accumulator. The following instructions will translate the value in the Accumulator to one of four values 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) ← ((A) + (DPTR))

MOVC A,@A + PC

Bytes:	1
Cycles:	2
Encoding:	`1 0 0 0` `0 0 1 1`
Operation:	MOVC (PC) ← (PC) + 1 (A) ← ((A) + (PC))

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

Function: Move External

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

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

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

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

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

MOVX   A,@R1

MOVX   @R0,A

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

MOVX   A,@Ri

Bytes: 1

Cycles: 2

Encoding: | 1 1 1 0 | 0 0 1 i |

Operation: MOVX
             (A) ← ((Ri))

MOVX   A,@DPTR

Bytes: 1

Cycles: 2

Encoding: | 1 1 1 0 | 0 0 0 0 |

Operation: MOVX
             (A) ← ((DPTR))

MOVX   @Ri,A

Bytes: 1

Cycles: 2

Encoding: | 1 1 1 1 | 0 0 1 i |

Operation: MOVX
             ((Ri)) ← (A)

MOVX   @DPTR,A

Bytes: 1

Cycles: 2

Encoding: | 1 1 1 1 | 0 0 0 0 |

Operation: MOVX
             (DPTR) ← (A)

MUL   AB

Function:	Multiply
Description:	MUL AB multiplies the unsigned eight-bit integers in the Accumulator and register B. The low-order byte of the sixteen-bit product is left in the Accumulator, and the high-order byte in B. If the product is greater than 255 (0FFH) the overflow flag is set; otherwise it is cleared. The carry flag is always cleared.
Example:	Originally the Accumulator holds the value 80 (50H). Register B holds the value 160 (0A0H). The instruction,

MUL   AB

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

Bytes:	1
Cycles:	4
Encoding:	1 0 1 0 | 0 1 0 0
Operation:	MUL $(A)_{7-0} \leftarrow (A) \times (B)$ $(B)_{15-8}$

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 $(PC) \leftarrow (PC) + 1$

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 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) ← (A) ∨ (Rn)

ORL   A,direct

Bytes:	2
Cycles:	1

Encoding: | 0 1 0 0 | 0 1 0 1 | | direct address |

Operation: ORL
(A) ← (A) ∨ (direct)

ORL   A,@Ri

Bytes:	1
Cycles:	1

Encoding: | 0 1 0 0 | 0 1 1 i |

Operation: ORL
(A) ← (A) ∨ ((Ri))

ORL   A,#data

Bytes:	2
Cycles:	1

Encoding: | 0 1 0 0 | 0 1 0 0 | | immediate data |

Operation: ORL
(A) ← (A) ∨ #data

ORL   direct,A

Bytes:	2
Cycles:	1

Encoding: | 0 1 0 0 | 0 0 1 0 | | direct address |

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

ORL   direct,#data

Bytes:	3
Cycles:	2

Encoding: | 0 1 0 0 | 0 0 1 1 | | direct addr. | | immediate data |

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

ORL   C,<src-bit>

Function:	Logical-OR for bit variables
Description:	Set the carry flag if the Boolean value is a logical 1; leave the carry in its current state otherwise . A slash ("/") preceding the operand in the assembly language indicates that the logical complement of the addressed bit is used as the source value, but the source bit itself is not affected. No other flags are affected.
Example:	Set the carry flag if and only if P1.0 = 1, ACC. 7 = 1, or OV = 0:

    MOV   C,P1.0   ;LOAD CARRY WITH INPUT PIN P10

    ORL   C,ACC.7   ;OR CARRY WITH THE ACC. BIT 7

    ORL   C,/OV   ;OR CARRY WITH THE INVERSE OF OV.

ORL   C,bit

Bytes:	2
Cycles:	2
Encoding:	`0 1 1 1` `0 0 1 0` `bit address`
Operation:	ORL (C) ← (C) ∨ (bit)

ORL   C,/bit

Bytes:	2
Cycles:	2
Encoding:	`1 0 1 0` `0 0 0 0` `bit address`
Operation:	ORL (C) ← (C) ∨ $\overline{(bit)}$

POP direct

Function:	Pop from stack.
Description:	The contents of the internal RAM location addressed by the Stack Pointer is read, and the Stack Pointer is decremented by one. The value read is then transferred to the directly addressed byte indicated. No flags are affected.
Example:	The Stack Pointer originally contains the value 32H, and internal RAM locations 30H through 32H contain the values 20H, 23H, and 01H, respectively. The instruction sequence,

POP   DPH

POP   DPL

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

POP   SP

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

Bytes:	2
Cycles:	2
Encoding:	`1 1 0 1` `0 0 0 0` `direct address`
Operation:	POP (direct) ← ((SP)) (SP) ← (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 an interrupt routine the Stack Pointer contains 09H. The Data Pointer holds the value 0123H. The instruction sequence,

PUSH   DPL

PUSH   DPH

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

Bytes:	2
Cycles:	2
Encoding:	`1 1 0 0` `0 0 0 0` `direct address`
Operation:	PUSH (SP) ← (SP) + 1 ((SP)) ← (direct)

RET

Function:	Return from subroutine
Description:	RET pops the high- and low-order bytes of the PC successively from the stack, decrementing the Stack Pointer by two. Program execution continues at the resulting address, generally the instruction immediately following an ACALL or LCALL. No flags are affected.
Example:	The Stack Pointer originally contains the value 0BH. Internal RAM locations 0AH and 0BH contain the values 23H and 01H, respectively. The instruction,

RET

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

Bytes:	1
Cycles:	2

Encoding:

0 0 1 0	0 0 1 0

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

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

$(A_n + 1) \leftarrow (An)$ $n = 0 - 6$
$(A0) \leftarrow (A7)$

RLC A

Function:	Rotate Accumulator Left through the Carry flag
Description:	The eight bits in the Accumulator and the carry flag are together rotated one bit to the left. Bit 7 moves into the carry flag; the original state of the carry flag moves into the bit 0 position. No other flags are affected.
Example:	The Accumulator holds the value 0C5H (11000101B), and the carry is zero. The instruction,

RLC A

leaves the Accumulator holding the value 8BH (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

$(An) \leftarrow (A_n + 1)$ $n = 0 - 6$

$(A7) \leftarrow (A0)$

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), 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

$(An) \leftarrow (An + 1)$ $n = 0 - 6$

$(A7) \leftarrow (C)$

$(C) \leftarrow (A0)$

SETB   <bit>

Function:	Set Bit
Description:	SETB sets the indicated bit to one. SETB can operate on the carry flag or any directly addressable bit. No other flags are affected.
Example:	The carry flag is cleared. Output Port 1 has been written with the value 34H (00110100B). The instructions,

SETB   C

SETB   P1.0

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

SETB   C

Bytes:	1
Cycles:	1

Encoding:

1 1 0 1	0 0 1 1

Operation: SETB
(C) ← 1

SETB   bit

Bytes:	2
Cycles:	1

Encoding:

1 1 0 1	0 0 1 0		bit address

Operation: SETB
(bit) ← 1

SJMP   rel

Function:	Short Jump
Description:	Program control branches unconditionally to the address indicated. The branch destination is computed by adding the signed displacement in the second instruction byte to the PC, after incrementing the PC twice. Therefore, the range of destinations allowed is from 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 a one-instruction infinite loop.)

Bytes:	2
Cycles:	2

Encoding:

1 0 0 0	0 0 0 0		rel. address

Operation: SJMP
$(PC) \leftarrow (PC) + 2$
$(PC) \leftarrow (PC) + rel$

SUBB A,<src-byte>

Function:	Subtract with borrow
Description:	SUBB subtracts the indicated variable and the carry flag together from the Accumulator, leaving the result in the Accumulator. SUBB sets the carry (borrow) flag if a borrow is needed for bit 7, and clears C otherwise. (If C was set *before* executing a SUBB instruction, this indicates that a borrow was needed for the previous step in a multiple precision subtraction, so the carry is subtracted from the Accumulator along with the source operand.) AC is set if a borrow is needed for bit 3, and cleared otherwise. OV is set if a borrow is needed into bit 6, but not into bit 7, or into bit 7, but not bit 6.
	When subtracting signed integers OV indicates a negative number produced when a negative value is subtracted from a positive value, or a positive result when a positive number is subtracted from a negative number.
	The source operand allows four addressing modes: register, direct, register-indirect, or immediate.
Example:	The Accumulator holds 0C9H (11001001B), register 2 holds 54H (01010100B), and the carry flag is set. The instruction,
	SUBB A,R2
	will leave the value 74H (01110100B) in the accumulator, with the carry flag and AC cleared but OV set.
	Notice that 0C9H minus 54H is 75H. The difference between this and the above result is due to the carry (borrow) flag being set before the operation. If the state of the carry is not known before starting a single or multiple-precision subtraction, it should be explicitly cleared by a CLR C instruction.

SUBB A,Rn

Bytes:	1
Cycles:	1
Encoding:	1 0 0 1 | 1 r r r
Operation:	SUBB (A) ← (A) − (C) − (Rn)

SUBB A,direct

Bytes:	2
Cycles:	1

Encoding:

1 0 0 1	0 1 0 1		direct address

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

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

XCH   A,<byte>

Function:	Exchange Accumulator with byte variable
Description:	XCH loads the Accumulator with the contents of the indicated variable, at the same time writing the original Accumulator contents to the indicated variable. The source/destination operand can use register, direct, or register-indirect addressing.
Example:	R0 contains the address 20H. The Accumulator holds the value 3FH (00111111B). Internal RAM location 20H holds the value 75H (01110101B). The instruction,

XCH   A,@R0

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

XCH   A,Rn

Bytes: 1

Cycles: 1

Encoding:

1 1 0 0	1 r r r

Operation: XCH
(A) $\rightleftarrows$ (Rn)

XCH   A,direct

Bytes: 2

Cycles: 1

Encoding:

1 1 0 0	0 1 0 1	direct address

Operation: XCH
(A) $\rightleftarrows$ (direct)

XCH   A,@Ri

Bytes: 1

Cycles: 1

Encoding:

1 1 0 0	0 1 1 i

Operation: XCH
(A) $\rightleftarrows$ ((Ri))

XCHD   A,@Ri

Function:	Exchange Digit
Description:	XCHD exchanges the low-order nibble of the Accumulator (bits 3-0), generally representing a hexadecimal or BCD digit, with that of the internal RAM location indirectly addressed by the specified register. The high-order nibbles (bits 7-4) of each register are not affected. No flags are affected.
Example:	R0 contains the address 20H. The Accumulator holds the value 36H (00110110B). Internal RAM location 20H holds the value 75H (01110101B). The instruction,

XCHD   A,@R0

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

Bytes:	1
Cycles:	1
Encoding:	$\boxed{1\ 1\ 0\ 1 \mid 0\ 1\ 1\ i}$
Operation:	XCHD $(A_{3-0}) \rightleftarrows ((Ri_{3-0}))$

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

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

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

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

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

XRL   A,R0

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

When the destination is a directly addressed byte, this instruction can complement combinations 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
   $(A) \leftarrow (A) \veebar (Rn)$

XRL   A,direct

   Bytes: 2

   Cycles: 1

   Encoding: | 0 1 1 0 | 0 1 0 1 | | direct address |

   Operation: XRL
   $(A) \leftarrow (A) \veebar (direct)$

XRL   A,@Ri

   Bytes: 1

   Cycles: 1

   Encoding: | 0 1 1 0 | 0 1 1 i |

   Operation: XRL
   $(A) \leftarrow (A) \veebar ((Ri))$

XRL   A,#data

   Bytes: 2

   Cycles: 1

   Encoding: | 0 1 1 0 | 0 1 0 0 | | immediate data |

   Operation: XRL
   $(A) \leftarrow (A) \veebar \#data$

XRL   direct,A

   Bytes: 2

   Cycles: 1

   Encoding: | 0 1 1 0 | 0 0 1 0 | | direct address |

   Operation: XRL
   $(direct) \leftarrow (direct) \veebar (A)$

XRL   direct,#data

Bytes:	3
Cycles:	2

Encoding: | 0 1 1 0 | 0 0 1 1 | | direct address | | immediate data |

Operation: XRL
(direct) ⟵ (direct) ∀ #data