THE 8051 INSTRUCTION SET

The 8031/8051 microcontroller name is Copyright by Intel®.
www.intel.com

TABLE OF CONTENTS

4

TABLE OF CONTENTS 2

THE 8051 INSTRUCTION SET

All commands in alphabetic order:

ACALL addr11	DIV AB	LJMP addr16	RETI
ADD A,<src	DJNZ <byte,<rel8	MOV <dest,<src	RL A
ADDC A,<src	INC <byte	MOV DPTR,#data16	RLC A
AJMP addr11	INC DPTR	MOV bit,bit	RR A
ANL <dest,<src	JB bit,rel8	MOVC A,@A+<base	RRC A
ANL <bit	JBC bit,rel8	MOVX <dest,<src	SETB bit
CJNE <dest,<src,rel8	JC rel8	MUL AB	SJMP rel8
CLR A	JMP @A+DPTR	NOP	SUBB A,<src
CLR bit	JNB bit,rel8	ORL <dest,<src	SWAP A
CPL A	JNC rel8	ORL C,bit	XCH A,<byte
CPL bit	JNZ rel8	POP direct	XCHD A,@Ri
DA A	JZ rel8	PUSH direct	XRL <dest,<src
DEC <byte	LCALL addr16	RET	

The standard 8051/52 CPU Registers.

Symbol	Name	Address	Reset-value
ACC *	Accumulator	E0	0000 0000 (00)
B *	B Register	F0	0000 0000 (00)
PSW *	Program Status Word	D0	0000 0000 (00)
SP	Stack Pointer	81	0000 0111 (07)
DPL	Low Byte Dptr	82	0000 0000 (00)
DPH	High Byte Dptr	83	0000 0000 (00)
P0 *	Port 0	80	1111 1111 (FF)
P1 *	Port 1	90	1111 1111 (FF)
P2 *	Port 2	A0	1111 1111 (FF)
P3 *	Port 3	B0	1111 1111 (FF)
IP *	Interrupt Priority Control	B8	xxx0 0000 (?0)
IE *	Interrupt Enable Control	A8	0xx0 0000 (?0)
TMOD	Timer/Counter Mode Control	89	0000 0000 (00)
TCON *	Timer/Counter Control	88	0000 0000 (00)
T2CON * ^	Timer/Counter 2 Control	C8	0000 0000 (00)
TH0	Timer/Counter 0 High Byte	8C	0000 0000 (00)
TL0	Timer/Counter 0 Low Byte	8A	0000 0000 (00)
TH1	Timer/Counter 1 High Byte	8D	0000 0000 (00)
TL1	Timer/Counter 1 Low Byte	8B	0000 0000 (00)
TH2 ^	Timer/Counter 1 High Byte	CD	0000 0000 (00)
TL2 ^	Timer/Counter 2 Low Byte	CC	0000 0000 (00)
RCAP2H ^	T/C 2 Capture Reg. High Byte	CB	0000 0000 (00)
RCAP2L ^	T/C 2 Capture Reg. Low Byte	CA	0000 0000 (00)
SCON *	Serial Control	98	0000 0000 (00)
SBUF	Serial Data Buffer	99	xxxx xxxx (??)
PCON	Power Control	87	0xxx 0000 (?0)

Description:
^ = 80C52
* = Bit addressable
x = Unknown state.

ARITHMETIC OPERATORS

Instruction	Description	Bytes	Periods	C	OV	AC
`ADD A, Rn`	Add register to ACC	1	12	x	x	x
`ADD A, direct`	Add direct byte to ACC	2	12	x	x	x
`ADD A, @Ri`	Add indirect RAM to ACC	1	12	x	x	x
`ADD A, #data`	Add immediate data to ACC	2	12	x	x	x
`ADDC A, Rn`	Add register to ACC with Carry	1	12	x	x	x
`ADDC A, direct`	Add direct byte to ACC with Carry	2	12	x	x	x
`ADDC A, @Ri`	Add indirect RAM to ACC with Carry	1	12	x	x	x
`ADDC A, #data`	Add immediate data to ACC with Carry	2	12	x	x	x
`SUBB A, Rn`	Subtract Register from ACC with borrow	1	12	x	x	x
`SUBB A, direct`	Subtract indirect RAM from ACC with borrow	2	12	x	x	x
`SUBB A, @Ri`	Subtract indirect RAM from ACC with borrow	1	12	x	x	x
`SUBB A, #data`	Subtract immediate data from ACC with borrow	2	12	x	x	x
`INC A`	Increment ACC	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 ACC	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	0	x	
`DIV AB`	Divide A by B	1	48	0	x	
`DA A`	Decimal Adjust ACC	1	12	x		

LOGICAL OPERATIONS

Instruction	Description	Bytes	Periods	C	OV	AC
`ANL A,Rn`	AND register to ACC	1	12			
`ANL A,direct`	AND direct byte to ACC	2	12			
`ANL A,@Ri`	AND indirect RAM to ACC	1	12			
`ANL A,#data`	AND immediate data to ACC	2	12			
`ANL direct,A`	AND ACC to direct byte	2	12			
`ANL direct,#data`	AND immediate data to direct byte	3	24			
`ORL A,Rn`	OR register to ACC	1	12			
`ORL A,direct`	OR direct byte to ACC	2	12			
`ORL A,@Ri`	OR indirect RAM to ACC	1	12			
`ORL A,#data`	OR immediate data to ACC	2	12			
`ORL direct,A`	OR ACC to direct byte	2	12			
`ORL direct,#data`	OR immediate data to direct byte	3	24			
`XRL A,Rn`	XOR register to ACC	1	12			
`XRL A,direct`	XOR direct byte to ACC	2	12			
`XRL A,@Ri`	XOR indirect RAM to ACC	1	12			
`XRL A,#data`	XOR immediate data to ACC	2	12			
`XRL direct,A`	XOR ACC to direct byte	2	12			
`XRL direct,#data`	XOR immediate data to direct byte	3	24			
`CLR A`	Clear the ACC	1	12			
`CPL A`	Complement the ACC	1	12			
`RL A`	Rotate the ACC left	1	12			

Instruction	Description	Bytes	Periods	C	OV	AC
RLC A	Rotate the ACC left through Carry	1	12	x		
RR A	Rotate the ACC right	1	12			
RRC A	Rotate the ACC right through Carry	1	12	x		
SWAP A	Swap nibbles in the ACC	1	12			

DATA TRANSFER

Instruction	Description	Bytes	Periods	C	OV	AC
MOV A,Rn	Move Register to ACC	1	12			
MOV A,direct	Move Direct byte to ACC	2	12			
MOV A,@Ri	Move Indirect byte to ACC	1	12			
MOV A,#data	Move Immediate data to ACC	2	12			
MOV Rn,A	Mov ACC 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 ACC to Direct byte	2	12			
MOV direct,Rn	Move Register to Direct byte	2	24			
MOV direct,direct	Move Direct byte to Direct byte	3	24			
MOV direct,@Ri	Mov Indirect RAM to Direct byte	3	24			
MOV direct,#data	Move Immediate data to Direct byte	3	24			
MOV @Ri,A	Move ACC to Indirect RAM	1	12			
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 datapointer with 16 bit constant	3	24			
MOVC A,@A+DPTR	Move code byte at ACC+DPTR to ACC	1	24			
MOVC A,@A+PC	Move code byte at ACC+PC to ACC	1	24			
MOVX A,@Ri	Move external RAM to ACC	1	24			
MOVX @Ri,A	Move ACC to external RAM	1	24			
MOVX A,@DPTR	Move external RAM to ACC	1	24			
MOVX @DPTR,A	Move ACC to external RAM	1	24			
PUSH direct	Push direct byte to stack	2	24			
POP direct	Pop direct byte from stack	2	24			
XCH A,Rn	Exchange register with ACC	1	12			
XCH A,direct	Exchange direct byte with ACC	2	12			
XCH A,@Ri	Exchange indirect RAM with ACC	1	12			
XCHD A,@Ri	Exchange low order digit indirect RAM with ACC	1	12			

BOOLEAN MANIPULATION

Instruction	Description	Bytes	Periods	C	OV	AC
CLR C	Clear carry flag	1	12	0		
CLR bit	Clear direct bit	2	12			
SETB C	Set carry flag	1	12	1		
SETB bit	Set direct bit	2	12			
CPL C	Complement carry flag	1	12	x		
CPL bit	Complement direct bit	2	12			
ANL C,bit	AND direct bit to carry	2	24	x		
ANL C,/bit	AND complement of direct bit to carry	2	24	x		
ORL C,bit	OR direct bit to carry	2	24	x		
ORL C,/bit	OR complement of direct bit to carry	2	24	x		
MOV C,bit	Move direct bit to carry	2	12	x		
MOV bit,C	Move carry to direct bit	2	24			
JC rel	Jump if carry is set	2	24			

`JNC rel`	Jump if carry is 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 and clear that bit	3	24			

ACALL addr11

Function:	Absolute Call within 2K block
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
Clocks:	24
Encoding:	a10 a9 a8 1:0 0 0 1 a7 a6 a5 a4:a3 a2 a1 a0
Operation:	(PC) « (PC) + 2 (SP) « (SP) + 1 [(SP)] « (PC7-0) (SP) « (SP) + 1 [(SP)] « (PC15-8) (PC10-0) « page address
Flags:	None

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

Function:	Add register byte to ACC
Bytes:	1
Clocks:	12
Encoding:	0 0 1 0:1 r r r
Operation:	(A) « (A) +(R_n)
Flags:	C AC OV

ADD A,direct

Function:	Add direct byte to ACC
Bytes:	2

Clocks:	12
Encoding:	0 0 1 0:0 1 0 1 a7 a6 a5 a4:a3 a2 a1 a0
Operation:	(A) « (A) + (direct)
Flags:	C AC OV

ADD A,@Ri

Function:	Add indirect RAM to ACC
Bytes:	1
Clocks:	12
Encoding:	0 0 1 0:0 1 0 i
Operation:	(A) « (A) + ((R_i))
Flags:	C AC OV

ADD A,#data

Function:	Add immediate data to ACC
Bytes:	2
Clocks:	12
Encoding:	0 0 1 0:0 1 0 0 d7 d6 d5 d4:d3 d2 d1 d0
Operation:	(A) « (A) + #data
Flags:	C AC OV

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

Function:	Add register to ACC with Carry
Bytes:	1
Clocks:	12
Encoding:	0 0 1 1:1 r r r
Operation:	(A) « (A) + (C) + (R_n)
Flags:	C AC OV

ADDC A,direct

Function:	Add direct byte to ACC with Carry
Bytes:	2
Clocks:	12
Encoding:	0 0 1 1:0 1 0 1 a7 a6 a5 a4:a3 a2 a1 a0
Operation:	(A) « (A) + (C) + (direct)
Flags:	C AC OV

ADDC A,@Ri

Function:	Add indirect RAM to ACC with Carry
Bytes:	1
Clocks:	12
Encoding:	0 0 1 1:0 1 1 i
Operation:	(A) « (A) + (C) + ((R_i))
Flags:	C AC OV

ADDC A,#data

Function:	Add immediate data to ACC with Carry
Bytes:	2
Clocks:	12
Encoding:	0 0 1 0:0 1 0 0 d7 d6 d5 d4:d3 d2 d1 d0
Operation:	(A) « (A) + (C) + #data
Flags:	C AC OV

AJMP address11

Function:	Absolute jump to address
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.
Bytes:	2
Clocks:	24
Encoding:	a10 a9 a8 0:0 0 0 1 a7 a6 a5 a4:a3 a2 a1 a0
Operation:	(PC) « (PC) + 2 (PC_{10-0}) « page address
Flags:	None

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

Function:	Logical AND register byte to ACC
Bytes:	1
Clocks:	12
Encoding:	0 1 0 1:1 r r r
Operation:	(A) « (A) ^ (R$_n$)
Flags:	

ANL A,direct

Function:	Logical AND direct byte to ACC
Bytes:	2
Clocks:	12
Encoding:	0 1 0 1:0 1 0 1 a7 a6 a5 a4:a3 a2 a1 a0
Operation:	(A) « (A) ^ (direct)
Flags:	

ANL A,@Ri

Function:	Logical AND indirect RAM to ACC
Bytes:	1
Clocks:	12
Encoding:	0 1 0 1:0 1 1 i
Operation:	(A) « (A) ^ ((R$_i$))
Flags:	

ANL A,#data

Function:	Logical AND immediate data to ACC
Bytes:	2
Clocks:	12
Encoding:	0 1 0 1:0 1 0 0 d7 d6 d5 d4:d3 d2 d1 d0
Operation:	(A) « (A) ^ #data
Flags:	

ANL direct,A

Function:	Logical AND ACC to direct data
Bytes:	2
Clocks:	12
Encoding:	0 1 0 1:0 0 1 0 d7 d6 d5 d4:d3 d2 d1 d0
Operation:	(direct) « (direct) ^ ACC
Flags:	

ANL direct,#data

Function:	Logical AND immediate data to direct data
Bytes:	3
Clocks:	24
Encoding:	0 1 0 1:0 0 1 1 a7 a6 a5 a4:a3 a2 a1 a0 d7 d6 d5 d4:d3 d2 d1 d0
Operation:	(direct) « (direct) ^ #data
Flags:	

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 ACC BIT 7
	ANL C,/OV ; AND WITH INVERSE OF OVERFLOW FLAG

ANL C,bit

Function:	Logical AND bit variable to carry
Bytes:	2
Clocks:	24
Encoding:	1 0 0 0:0 0 1 0 b7 b6 b5 b4:b3 b2 b1 b0
Operation:	(C) « (C) ^ (bit)
Flags:	C

ANL C,/bit

Function:	Logical AND bit variable to carry
Bytes:	2
Clocks:	24
Encoding:	1 0 1 1:0 0 0 0 b7 b6 b5 b4:b3 b2 b1 b0
Operation:	(C) « (C) ^ (bit)
Flags:	C

CJNE <dest-byte>,<src-byte>,rel8

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,rel8

Function:	Compare direct byte to ACC, Jump if Not Equal
Bytes:	3
Clocks:	24
Encoding:	1 0 1 1:0 1 0 1 a7 a6 a5 a4:a3 a2 a1 a0 r7 r6 r5 r4:r3 r2 r1 r0
Operation:	(PC) « (PC) + 3
	IF (A) <> (direct)
	THEN
	(PC) « (PC) + relative offset

	IF (A) < (direct)
	THEN
	(C) « 1
	ELSE
	(C) « 0

Flags:	C

CJNE A,#data,rel8

Function:	Compare immediate data to ACC, Jump if Not Equal
Bytes:	3
Clocks:	24
Encoding:	1 0 1 1:0 1 0 0 a7 a6 a5 a4:a3 a2 a1 a0 r7 r6 r5 r4:r3 r2 r1 r0

Operation:	(PC) « (PC) + 3
	IF (A) <> data
	THEN
	(PC) « (PC) + relative offset
	IF (A) < data
	THEN
	(C) « 1
	ELSE
	(C) « 0

Flags:	C

CJNE Rn,#data,rel8

Function:	Compare immediate data to register, Jump if Not Equal
Bytes:	3
Clocks:	24
Encoding:	1 0 1 1:1 r r r a7 a6 a5 a4:a3 a2 a1 a0 r7 r6 r5 r4:r3 r2 r1 r0

Operation:	(PC) « (PC) + 3
	IF (Rn) <> data
	THEN
	(PC) « (PC) + relative offset
	IF (Rn) < data
	THEN
	(C) « 1
	ELSE
	(C) « 0

Flags:	C

CJNE @Ri,#data,rel8

Function:	Compare immediate data to register, Jump if Not Equal
Bytes:	3
Clocks:	24
Encoding:	1 0 1 1:0 1 1 i a7 a6 a5 a4:a3 a2 a1 a0 r7 r6 r5 r4:r3 r2 r1 r0

Operation:	(PC) « (PC) + 3
	IF ((Ri)) <> data
	THEN
	(PC) « (PC) + relative offset
	IF ((Ri)) < data
	THEN
	(C) « 1
	ELSE
	(C) « 0

Flags:	C

CLR A

Function:	Clears the ACC
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
Clocks:	12
Encoding:	1 1 1 1:0 1 0 0
Operation:	(A) « 0
Flags:	None

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

Function:	Clears the carry
Bytes:	1
Clocks:	12
Encoding:	1 1 0 0:0 0 1 1
Operation:	(C) « 0
Flags:	C

CLR bit

Function:	Clears bit
Bytes:	2
Clocks:	12
Encoding:	1 1 0 0:0 0 1 0 b7 b6 b5 b4:b3 b2 b1 b0
Operation:	(bit) « 0
Flags:	

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
Clocks:	12
Encoding:	1 1 1 1:0 1 0 0
Operation:	(A) « (A)
Flags:	None

CPL bit

Function:	Complement bit
Description:	The bit variable specified is complemented. A bit which had been a one is changed to zero and vice-versa. No other flags are affected. CLR can operate on the `carry` or any `directly` addressable bit. Note: When this instruction is used to modify an output pin, the value used as the original data will be read from the output data latch, not the input pin.
Example:	Port 1 has previously been written with 5DH (01011101B). The instruction sequence. **CPL P1.1** **CPL P1.2** will leave the port set to 5BH (01011011B).

CPL C

Function:	Complements the carry
Bytes:	1
Clocks:	12
Encoding:	1 0 1 1:0 0 1 1
Operation:	(C) « ~~(C)~~
Flags:	C

CPL bit

Function:	Complements selected bit
Bytes:	2
Clocks:	12
Encoding:	1 0 1 1:0 0 1 0 b7 b6 b5 b4:b3 b2 b1 b0
Operation:	(bit) « ~~(bit)~~
Flags:	

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, **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 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
Clocks:	12
Encoding:	1 1 0 1:0 1 0 0
Operation:	IF $[[(A3 - 0) > 9]$ V $[(AC) = 1]]$ THEN $(A3 - 0) \ll (A3 - 0) + 6$ AND IF $[[(A7 - 4) > 9]$ V $[(C) = 1]]$ THEN $(A7 - 4) \ll (A7 - 4) + 6$
Flags:	C

DEC byte

Function:	Decrement
Description:	The variable indicated is decremented by one. 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 internal RAM locations 7EH and 7FH to 0FFH and 3FH.

DEC A

Function:	Decrement ACC by one
Bytes:	1
Clocks:	12
Encoding:	0 0 0 1:0 1 0 0
Operation:	(A) « (A) - 1
Flags:	

DEC Rn

Function:	Decrement register by one
Bytes:	1
Clocks:	12
Encoding:	0 0 0 1:1 r r r
Operation:	(Rn) « (Rn) - 1
Flags:	

DEC direct

Function:	Decrement direct byte by one
Bytes:	1
Clocks:	12
Encoding:	0 0 0 1:0 1 0 1
Operation:	(direct) « (direct) - 1
Flags:	

DEC @Ri

Function:	Decrement indirect RAM by one
Bytes:	1
Clocks:	12
Encoding:	0 0 0 1:0 1 1 i
Operation:	((Ri)) « ((Ri)) - 1
Flags:	

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 $251 = (13 \times 18) + 17$. Carry and OV will both be cleared.
Bytes:	1
Clocks:	48
Encoding:	1 0 0 0:0 1 0 0
Operation:	(A)15-8 « (A)/(B) (B)7-0
Flags:	OV, C

DJNZ <byte>,rel8

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 alter the pin.

DJNZ Rn,rel8

Function:	Decrement register, Jump if Not Zero
Bytes:	2
Clocks:	24
Encoding:	1 1 0 1:1 r r r r7 r6 r5 r4:r3 r2 r1 r0
Operation:	(PC) « (PC) + 2
	(Rn) « (Rn) - 1
	IF (RN) > 0 or (Rn) < 0
	THEN
	(PC) « (PC) + rel8
Flags:	

DJNZ direct,rel8

Function:	Decrement direct byte, Jump if Not Zero
Bytes:	3
Clocks:	24
Encoding:	1 1 0 1:0 1 0 1 a7 a6 a5 a4:a3 a2 a1 a0 r7 r6 r5 r4:r3 r2 r1 r0
Operation:	(PC) « (PC) + 2
	(direct) « (direct) - 1
	IF (direct) > 0 or (direct) < 0
	THEN
	(PC) « (PC) + rel8
Flags:	

INC byte

Function:	Decrement
Description:	INC increments the indicated variable by 1.
	An original value of 0FFH will overflow to 00H. No flags are affected.
	Four 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 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

Function:	Increment ACC by one
Bytes:	1
Clocks:	12
Encoding:	0 0 0 0:0 1 0 0
Operation:	(A) « (A) + 1
Flags:	

INC Rn

Function:	Increment register by one
Bytes:	1
Clocks:	12
Encoding:	0 0 0 0:1 r r r
Operation:	(Rn) « (Rn) + 1
Flags:	

INC direct

Function:	Increment direct byte by one
Bytes:	1
Clocks:	12
Encoding:	0 0 0 0:0 1 0 1
Operation:	(direct) « (direct) + 1
Flags:	

INC @Ri

Function:	Increment indirect RAM by one
Bytes:	1
Clocks:	12
Encoding:	0 0 0 0:0 1 1 i
Operation:	((Ri)) « ((Ri)) + 1
Flags:	

INC DPTR

Function:	Increment Data Pointer
Description:	Increment the 16-bit data pointer by 1.A 16-bit increment (modulo2 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
Clocks:	24
Encoding:	1 0 1 0:0 0 1 1
Operation:	(DPTR) « (DPTR) + 1
Flags:	None

JB bit,rel8

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. 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, LABEL1** **JB ACC.2, LABEL2**

	will cause program execution to branch to the instruction at label LABEL 2.
Bytes:	3
Clocks:	24
Encoding:	0 0 1 0:0 0 0 0 b7 b6 b5 b4:b3 b2 b1 b0 r7 r6 r5 r4:r3 r2 r1 r0
Operation:	(PC) « (PC) + 3 IF (bit) = 1 THEN (PC) « (PC) + rel8
Flags:	None

JBC bit,rel8

Function:	Jump if Bit 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
Clocks:	24
Encoding:	0 0 0 1:0 0 0 0 b7 b6 b5 b4:b3 b2 b1 b0 r7 r6 r5 r4:r3 r2 r1 r0
Operation:	(PC) « (PC) + 3 IF (bit) = 1 THEN (bit) « 0 (PC) « (PC) + rel8
Flags:	None

JC bit,rel8

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 LABEL2** will set the carry and cause program execution to continue at the instruction identified by the label LABEL2.
Bytes:	2
Clocks:	24
Encoding:	0 1 0 0:0 0 0 0 r7 r6 r5 r4:r3 r2 r1 r0
Operation:	(PC) « (PC) + 2 IF (C) = 1 THEN (PC) « (PC) + rel8
Flags:	None

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 (modulo2 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
Clocks:	24
Encoding:	0 1 1 1:0 0 1 1
Operation:	(PC) « ((A)) + (DPTR)
Flags:	None

JNB bit,rel8

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
Clocks:	24
Encoding:	0 0 1 1:0 0 0 0 b7 b6 b5 b4:b3 b2 b1 b0 r7 r6 r5 r4:r3 r2 r1 r0
Operation:	(PC) « (PC) + 3 IF (bit) = 0 THEN (PC) « (PC) + rel
Flags:	None

JNC rel8

Function:	Jump if Carry flag 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
Clocks:	24
Encoding:	0 1 0 1:0 0 0 0 r7 r6 r5 r4:r3 r2 r1 r0
Operation:	(PC) « (PC) + 2 IF (C) = 0 THEN (PC) « (PC) + rel
Flags:	None

JNZ rel8

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
Clocks:	24
Encoding:	0 1 1 1:0 0 0 0 r7 r6 r5 r4:r3 r2 r1 r0
Operation:	(PC) « (PC) + 2 IF (A) <> 0 THEN (PC) « (PC) + rel
Flags:	None

JZ rel8

Function:	Jump if Accumulator is 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
Clocks:	24
Encoding:	0 1 1 1:0 0 0 0 r7 r6 r5 r4:r3 r2 r1 r0
Operation:	(PC) « (PC) + 2 IF (A) = 0 THEN (PC) « (PC) + rel
Flags:	None

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
Clocks:	24
Encoding:	0 0 0 1:0 0 1 0 a15 a14 a13 a12:a11 a10 a9 a8 a7 a6 a5 a4:a3 a2 a1 a0
Operation:	(PC) « (PC) + 3 (SP) « (SP) + 1 ((SP)) « (PC7-0) (SP) « (SP) + 1 ((SP)) « (PC15-8) (PC) « addr15-0
Flags:	None

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
Clocks:	24
Encoding:	0 0 0 0:0 0 1 0 a15 a14 a13 a12:a11 a10 a9 a8 a7 a6 a5 a4:a3 a2 a1 a0
Operation:	(PC) « addr15-0
Flags:	None

MOV <dest>,<source>

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 combinations of source and destination addressing modes are allowed:
	`MOV A,Rn MOV A,direct MOV A,@Ri MOV A,#data`
	`MOV Rn,A MOV Rn,direct MOV Rn,#data`
	`MOV direct,A MOV direct,Rn MOV direct,direct MOV direct,@Ri MOV direct,#data`
	`MOV @Ri,A MOV @Ri,direct MOV @Ri,#data`
	Note: Mov A,ACC is not a valid instruction.

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

Function:	Move register to ACC
Bytes:	1
Clocks:	12
Encoding:	1 1 1 0:1 r r r
Operation:	(A) « (Rn)
Flags:	

MOV A,direct

Function:	Move direct byte to ACC
Bytes:	2
Clocks:	12
Encoding:	1 1 1 0:0 1 0 1 a7 a6 a5 a4:a3 a2 a1 a0
Operation:	(A) « (direct)
Flags:	

MOV A, @Ri

Function:	Move indirect ram to ACC
Bytes:	1
Clocks:	12
Encoding:	1 1 1 0:0 1 1 i
Operation:	(A) « ((Ri))
Flags:	

MOV A,#data

Function:	Move immediate data to ACC
Bytes:	2
Clocks:	12
Encoding:	0 1 1 1:0 1 0 0 d7 d6 d5 d4:d3 d2 d1 d0
Operation:	(A) « #data
Flags:	

MOV Rn,A

Function:	Move ACC to register
Bytes:	1
Clocks:	12
Encoding:	1 1 1 1:1 r r r
Operation:	(Rn) « (A)
Flags:	

MOV Rn,direct

Function:	Move direct byte to register
Bytes:	2
Clocks:	24

Encoding:	1 0 1 0:1 r r r a7 a6 a5 a4:a3 a2 a1 a0
Operation:	(Rn) « (direct)
Flags:	

MOV Rn,#data

Function:	Move immediate data to register
Bytes:	2
Clocks:	12
Encoding:	0 1 1 1:1 r r r d7 d6 d5 d4:d3 d2 d1 d0
Operation:	(Rn) « #data
Flags:	

MOV direct,A

Function:	Move ACC to direct byte
Bytes:	2
Clocks:	12
Encoding:	1 1 1 1:0 1 0 1 a7 a6 a5 a4:a3 a2 a1 a0
Operation:	(direct) « (A)
Flags:	

MOV direct,Rn

Function:	Move register to direct byte
Bytes:	2
Clocks:	24
Encoding:	1 0 0 0:1 r r r a7 a6 a5 a4:a3 a2 a1 a0
Operation:	(direct) « (Rn)
Flags:	

MOV direct,direct

Function:	Move direct byte to direct byte
Bytes:	3
Clocks:	24
Encoding:	1 0 0 0:0 1 0 1 a7 a6 a5 a4:a3 a2 a1 a0(src) a7 a6 a5 a4:a3 a2 a1 a0(dest)
Operation:	(direct) « (direct)
Flags:	

MOV direct,@Ri

Function:	Move indirect ram to direct byte
Bytes:	2
Clocks:	24
Encoding:	1 0 0 0:0 1 1 i a7 a6 a5 a4:a3 a2 a1 a0
Operation:	(direct) « ((Ri))
Flags:	

MOV direct,#data

Function:	Move immediate data to direct byte
Bytes:	3
Clocks:	24
Encoding:	1 0 0 0:0 1 0 1 a7 a6 a5 a4:a3 a2 a1 a0 d7 d6 d5 d4:d3 d2 d1 d0
Operation:	(direct) « #data
Flags:	

MOV @Ri,A

Function:	Move ACC to indirect ram
Bytes:	1
Clocks:	12
Encoding:	1 1 1 1:0 1 1 i
Operation:	((Ri)) « (A)
Flags:	

MOV @Ri,direct

Function:	Move direct byte to indirect ram
Bytes:	2
Clocks:	24
Encoding:	1 0 1 0:0 1 1 i a7 a6 a5 a4:a3 a2 a1 a0
Operation:	((Ri)) « (direct)
Flags:	

MOV @Ri,#data

Function:	Move immediate data to indirect ram
Bytes:	2
Clocks:	12
Encoding:	0 1 1 1:0 1 1 i d7 d6 d5 d4:d3 d2 d1 d0
Operation:	((Ri)) « #data
Flags:	

MOV <dest-bit>,<source-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

Function:	Move bit data to Carry
Bytes:	2
Clocks:	12
Encoding:	1 0 1 0:0 0 1 0 b7 b6 b5 b4:b3 b2 b1 b0
Operation:	(C) « (bit)
Flags:	

MOV A,direct

Function:	Move Carry to bit data
Bytes:	2
Clocks:	24
Encoding:	1 0 0 1:0 0 1 0 b7 b6 b5 b4:b3 b2 b1 b0
Operation:	(bit) « (C)
Flags:	

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
Clocks:	24
Encoding:	1 0 0 1:0 0 0 0 d15 d14 d13 d12:d11 d10 d9 d8 d7 d6 d5 d4:d3 d2 d1 d0
Operation:	(DPH) « #data15-8 (DPL) « #data7-0
Flags:	None

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** **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.
Flags:	

MOVC A,@A+DPTR

Function:	Move Code byte at ACC + DPTR to ACC
Bytes:	1
Clocks:	24
Encoding:	1 0 0 1:0 0 1 1
Operation:	(A) « ((A) + (DPTR))
Flags:	None

MOVC A,@A+PC

Function:	Move Code byte at ACC + PC to ACC
Bytes:	1

Clocks:	24
Encoding:	1 0 0 0:0 0 1 1
Operation:	(PC) « (PC) + 1 (A) « ((A) + (PC))
Flags:	None

MOVX A,@A+<base-reg>

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.
Flags:	

MOVX A,@Ri

Function:	
Bytes:	1
Clocks:	24
Encoding:	1 1 1 0:0 0 1 i
Operation:	(A) « ((Ri))
Flags:	None

MOVX A,@DPTR

Function:	Move external byte at DPTR to ACC
Bytes:	1
Clocks:	24
Encoding:	1 0 0 0:0 0 1 1
Operation:	(A) « ((DPTR))
Flags:	None

MOVX @Ri,A

Function:	
Bytes:	1
Clocks:	24
Encoding:	1 1 1 1:0 0 1 i
Operation:	((Ri)) « (A)
Flags:	None

MOVX @DPTR,A

Function:	Move ACC to external byte at DPTR
Bytes:	1
Clocks:	24
Encoding:	1 1 1 1:0 0 0 0
Operation:	((DPTR)) « (A)
Flags:	None

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
Clocks:	48
Encoding:	1 0 1 0:0 1 0 0
Operation:	(PC) « (PC) + 1
Flags:	OV C

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** **SETB P2.7**
Bytes:	1
Clocks:	12
Encoding:	0 0 0 0:0 0 0 0
Operation:	(PC) « (PC) + 1
Flags:	None

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. 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. No flags are affected.
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

Function:	Logical OR register byte to ACC
Bytes:	1
Clocks:	12
Encoding:	0 1 0 0:1 r r r
Operation:	(A) « (A) v (R$_n$)
Flags:	

ORL A,direct

Function:	Logical OR direct byte to ACC
Bytes:	2
Clocks:	12
Encoding:	0 1 0 0:0 1 0 1 a7 a6 a5 a4:a3 a2 a1 a0
Operation:	(A) « (A) v (direct)
Flags:	

ORL A,@Ri

Function:	Logical OR indirect RAM to ACC
Bytes:	1
Clocks:	12
Encoding:	0 1 0 0:0 1 1 i
Operation:	(A) « (A) v ((R$_i$))
Flags:	

ORL A,#data

Function:	Logical OR immediate data to ACC
Bytes:	2
Clocks:	12
Encoding:	0 1 0 0:0 1 0 0 d7 d6 d5 d4:d3 d2 d1 d0
Operation:	(A) « (A) v #data
Flags:	

ORL direct,A

Function:	Logical OR ACC to direct data
Bytes:	2
Clocks:	12
Encoding:	0 1 0 0:0 0 1 0 d7 d6 d5 d4:d3 d2 d1 d0
Operation:	(direct) « (direct) v ACC
Flags:	

ORL direct,#data

Function:	Logical OR immediate data to direct data
Bytes:	3
Clocks:	24
Encoding:	0 1 0 0:0 0 1 1 a7 a6 a5 a4:a3 a2 a1 a0 d7 d6 d5 d4:d3 d2 d1 d0
Operation:	(direct) « (direct) v #data
Flags:	

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 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 P1.0 **ORL C, ACC.7 ;** OR CARRY WITH THE ACC BIT 7 **ORL C,/OV ;** OR CARRY WITH THE INVERSE OF OV

ORL C,bit

Function:	Logical OR bit variable to carry
Bytes:	2
Clocks:	24
Encoding:	0 1 1 1:0 0 1 0 b7 b6 b5 b4:b3 b2 b1 b0
Operation:	(C) « (C) v (bit)
Flags:	C

ORL C,/bit

Function:	Logical OR complement of bit variable to carry
Bytes:	2
Clocks:	24
Encoding:	1 0 1 0:0 0 0 0 b7 b6 b5 b4:b3 b2 b1 b0
Operation:	(C) « (C) v (bit)
Flags:	C

POP direct

Function:	Pop from the 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
Clocks:	24
Encoding:	1 1 0 1:0 0 0 0 a7 a6 a5 a4:a3 a2 a1 a0
Operation:	(direct) « ((SP))
	(SP) « (SP) -1
Flags:	None

PUSH direct

Function:	Push onto the stack
Description:	The Stack Pointer is incremented by one.
	The contents for 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
Clocks:	24
Encoding:	1 1 0 0:0 0 0 0 a7 a6 a5 a4:a3 a2 a1 a0
Operation:	(SP) « (SP) + 1
	((SP)) « (direct)
Flags:	None

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
Clocks:	24
Encoding:	0 0 1 0:0 0 1 0
Operation:	(PC15-8) « ((SP))
	(SP) « (SP) - 1
	(PC7-0) « ((SP))
	(SP) « (SP) - 1
Flags:	None

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
Clocks:	24
Encoding:	0 0 1 1:0 0 1 0
Operation:	(PC15-8) « ((SP)) (SP) « (SP) - 1 (PC7-0) « ((SP)) (SP) « (SP) - 1
Flags:	None

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 (10001011B) with the carry unaffected.
Bytes:	1
Clocks:	12
Encoding:	0 0 1 0:0 0 1 1
Operation:	(An + 1) « (An) n = 0-6 (A0) « (A7)
Flags:	None

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
Clocks:	12
Encoding:	0 0 1 1:0 0 1 1
Operation:	(An + 1) « (An) n = 0-6 (A0) « (C) (C) « (A7)

Flags:	C

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
Clocks:	12
Encoding:	0 0 0 0:0 0 1 1
Operation:	(An) « (An + 1) n = 0-6 (A7) « (A0)
Flags:	none

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
Clocks:	12
Encoding:	0 0 0 1:0 0 1 1
Operation:	(An) « (An + 1) n = 0-6 (A7) « (C) (C) « (A0)
Flags:	C

SETB <bit>

Function:	Setb 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

Function:	Set the Carry
Bytes:	1
Clocks:	12
Encoding:	1 1 0 1:0 0 1 1
Operation:	(C) « 1
Flags:	C

Function:	Set bit variable
Bytes:	2
Clocks:	12
Encoding:	1 1 0 1:0 0 1 0 b7 b6 b5 b4:b3 b2 b1 b0
Operation:	(bit) « 1
Flags:	

SJMP rel8

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
Clocks:	24
Encoding:	1 0 0 0:0 0 0 0 r7 r6 r5 r4:r3 r2 r1 r0
Operation:	(PC) « (PC) + 2 (PC) « (PC) + rel
Flags:	

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 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. Four source operand addressing mode are allowed: `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 CLR C instruction.

SUBB A,Rn

Function:	Subtract register to ACC with borrow
Bytes:	1
Clocks:	12
Encoding:	1 0 0 1:1 r r r
Operation:	(A) « (A) - (C) - (R$_n$)
Flags:	C AC OV

SUBB A,direct

Function:	Subtract direct byte to ACC with borrow
Bytes:	2
Clocks:	12
Encoding:	1 0 0 1:0 1 0 1 a7 a6 a5 a4:a3 a2 a1 a0
Operation:	(A) « (A) - (C) - (direct)
Flags:	C AC OV

SUBB A,@Ri

Function:	Subtract indirect RAM to ACC with borrow
Bytes:	1
Clocks:	12
Encoding:	1 0 0 1:0 1 1 i
Operation:	(A) « (A) - (C) - ((R$_i$))
Flags:	C AC OV

SUBB A,#data

Function:	Subtract immediate data to ACC with borrow
Bytes:	2
Clocks:	12
Encoding:	1 0 0 1:0 1 0 0 d7 d6 d5 d4:d3 d2 d1 d0
Operation:	(A) « (A) - (C) - #data
Flags:	C AC OV

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 flags are affected.
Example:	The Accumulator holds the value 0C5H (11000101B). The instruction, **SWAP A** leave the Accumulator holding the value 5CH (01011100B).
Bytes:	1
Clocks:	12
Encoding:	1 1 0 0:0 1 0 0
Operation:	(A3-0) «» (A7-4)
Flags:	None

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

Function:	Exchange ACC with register
Bytes:	1
Clocks:	12
Encoding:	1 1 0 0:1 r r r
Operation:	(A) «» (Rn)
Flags:	

XCH A,direct

Function:	Exchange ACC with direct byte
Bytes:	2
Clocks:	12
Encoding:	1 1 0 0:0 1 0 1 a7 a6 a5 a4:a3 a2 a1 a0
Operation:	(A) «» (direct)
Flags:	

XCH A,@Ri

Function:	Exchange ACC with indirect RAM
Bytes:	1
Clocks:	12
Encoding:	1 1 0 0:0 1 1 i
Operation:	(A) «» ((Ri))
Flags:	

XCHD A,@Ri

Function:	Exchange digit
Description:	XCHD exchanges the low order nibble of the ACC (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 ACC holds the value 36H (00110110b). Internal RAM location 20H holds the value 75H (01110101b). The instruction, **XHCD A,@R0** will leave RAM location 20H holding the value 76H (01110110b) and 35H (00110101B) in the ACC.
Bytes:	1
Clocks:	12
Encoding:	1 1 0 1:0 1 1 i
Operation:	(A3-0) «» ((Ri3-0))
Flags:	

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

Function:	Logical Exclusive OR for byte variables
Description:	XRL performs the bitwise logical Exclusive OR 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 AAH (10101010B) then the instruction, **XRL A, R0** will leave 69H (01101001B) in the Accumulator. When the destination is a directly addressed byte, this instruction can complement 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, **XRL P1, #00110001B** will complement bits 5, 4, and 0 of output port 1.

XRL A,Rn

Function:	Logical Exclusive OR register byte to ACC
Bytes:	1
Clocks:	12
Encoding:	0 1 1 0:1 r r r
Operation:	(A) « (A) XOR (R$_n$)
Flags:	

XRL A,direct

Function:	Logical Exclusive OR direct byte to ACC
Bytes:	2
Clocks:	12
Encoding:	0 1 1 0:0 1 0 1 a7 a6 a5 a4:a3 a2 a1 a0
Operation:	(A) « (A) XOR (direct)
Flags:	

XRL A,@Ri

Function:	Logical Exclusive OR indirect RAM to ACC
Bytes:	1
Clocks:	12
Encoding:	0 1 1 0:0 1 1 i
Operation:	(A) « (A) XOR ((R$_i$))
Flags:	

XRL A,#data

Function:	Logical Exclusive OR immediate data to ACC
Bytes:	2
Clocks:	12
Encoding:	0 1 1 0:0 1 0 0 d7 d6 d5 d4:d3 d2 d1 d0
Operation:	(A) « (A) XOR #data
Flags:	

XRL direct,A

Function:	Logical Exclusive OR ACC to direct data
Bytes:	2
Clocks:	12
Encoding:	0 1 1 0:0 0 1 0 d7 d6 d5 d4:d3 d2 d1 d0
Operation:	(direct) « (direct) XOR ACC
Flags:	

ANL direct,#data

Function:	Logical Exclusive OR immediate data to direct data
Bytes:	3
Clocks:	24
Encoding:	0 1 1 0:0 0 1 1 a7 a6 a5 a4:a3 a2 a1 a0 d7 d6 d5 d4:d3 d2 d1 d0
Operation:	(direct) « (direct) XOR #data
Flags:	

Function:	Logical Exclusive OR ACC to direct data
Bytes:	2
Clocks:	12
Operation:	(direct) « (direct) XOR ACC