8051 Instruction Set

Instructions by opcode

	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0a	0x0b	0x0c	0x0d	0x0e	0x0f
0x00	NOP	AJMP	LJMP	RR	INC	INC	INC	INC	INC	INC	INC	INC	INC	INC	INC	INC
0x10	JBC	ACALL	LCALL	RRC	DEC	DEC	DEC	DEC	DEC	DEC	DEC	DEC	DEC	DEC	DEC	DEC
0x20	JB	AJMP	RET	RL	ADD	ADD	ADD	ADD	ADD	ADD	ADD	ADD	ADD	ADD	ADD	ADD
0x30	JNB	ACALL	RETI	RLC	ADDC	ADDC	ADDC	ADDC	ADDC	ADDC	ADDC	ADDC	ADDC	ADDC	ADDC	ADDC
0x40	JC	AJMP	ORL	ORL	ORL	ORL	ORL	ORL	ORL	ORL	ORL	ORL	ORL	ORL	ORL	ORL
0x50	JNC	ACALL	ANL	ANL	ANL	ANL	ANL	ANL	ANL	ANL	ANL	ANL	ANL	ANL	ANL	ANL
0x60	JZ	AJMP	XRL	XRL	XRL	XRL	XRL	XRL	XRL	XRL	XRL	XRL	XRL	XRL	XRL	XRL
0x70	JNZ	ACALL	ORL	JMP	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV
0x80	SJMP	AJMP	ANL	MOVC	DIV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV
0x90	MOV	ACALL	MOV	MOVC	SUBB	SUBB	SUBB	SUBB	SUBB	SUBB	SUBB	SUBB	SUBB	SUBB	SUBB	SUBB
0xa0	ORL	AJMP	MOV	INC	MUL	?	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV
0xb0	ANL	ACALL	CPL	CPL	CJNE	CJNE	CJNE	CJNE	CJNE	CJNE	CJNE	CJNE	CJNE	CJNE	CJNE	CJNE
0xc0	PUSH	AJMP	CLR	CLR	SWAP	XCH	XCH	XCH	XCH	XCH	XCH	XCH	XCH	XCH	XCH	XCH
0xd0	POP	ACALL	SETB	SETB	DA	DJNZ	XCHD	XCHD	DJNZ	DJNZ	DJNZ	DJNZ	DJNZ	DJNZ	DJNZ	DJNZ
0xe0	MOVX	AJMP	MOVX	MOVX	CLR	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV
0xf0	MOVX	ACALL	MOVX	MOVX	CPL	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV	MOV

Alphabetical List of Instructions

8051 Instruction Set: ACALL

Operation: ACALL
Function: Absolute Call Within 2K Block
Syntax: ACALL *code address*

Instructions	OpCode	Bytes	Flags
ACALL *page0*	0x11	2	None
ACALL *page1*	0x31	2	None
ACALL *page2*	0x51	2	None
ACALL *page3*	0x71	2	None
ACALL *page4*	0x91	2	None
ACALL *page5*	0xB1	2	None
ACALL *page6*	0xD1	2	None
ACALL *page7*	0xF1	2	None

Description: ACALL unconditionally calls a subroutine at the indicated *code address*. ACALL pushes the address of the instruction that follows ACALL onto the stack, least-significant-byte first, most-significant-byte second. The Program Counter is then updated so that program execution continues at the indicated address.

The new value for the Program Counter is calculated by replacing the least-significant-byte of the Program Counter with the second byte of the ACALL instruction, and replacing bits 0-2 of the most-significant-byte of the Program Counter with 3 bits that indicate the page. Bits 3-7 of the most-significant-byte of the Program Counter remain unchaged.

Since only 11 bits of the Program Counter are affected by ACALL, calls may only be made to routines located within the same 2k block as the first byte that follows ACALL.

See Also: LCALL, RET

8051 Instruction Set: ADD

Operation: ADD, ADDC
Function: Add Accumulator, Add Accumulator With Carry
Syntax: ADD A,*operand*
ADDC A,*operand*

Instructions	OpCode	Bytes	Flags
ADD A,*#data*	0x24	2	C, AC, OV

Instructions	OpCode	Bytes	Flags
ADDC A,*#data*	0x34	2	C, AC, OV

ADD A,*iram addr*	0x25	2	C, AC, OV	ADDC A,*iram addr*	0x35	2	C, AC, OV
ADD A,@R0	0x26	1	C, AC, OV	ADDC A,@R0	0x36	1	C, AC, OV
ADD A,@R1	0x27	1	C, AC, OV	ADDC A,@R1	0x37	1	C, AC, OV
ADD A,R0	0x28	1	C, AC, OV	ADDC A,R0	0x38	1	C, AC, OV
ADD A,R1	0x29	1	C, AC, OV	ADDC A,R1	0x39	1	C, AC, OV
ADD A,R2	0x2A	1	C, AC, OV	ADDC A,R2	0x3A	1	C, AC, OV
ADD A,R3	0x2B	1	C, AC, OV	ADDC A,R3	0x3B	1	C, AC, OV
ADD A,R4	0x2C	1	C, AC, OV	ADDC A,R4	0x3C	1	C, AC, OV
ADD A,R5	0x2D	1	C, AC, OV	ADDC A,R5	0x3D	1	C, AC, OV
ADD A,R6	0x2E	1	C, AC, OV	ADDC A,R6	0x3E	1	C, AC, OV
ADD A,R7	0x2F	1	C, AC, OV	ADDC A,R7	0x3F	1	C, AC, OV

Description: Description: ADD and ADDC both add the value *operand* to the value of the Accumulator, leaving the resulting value in the Accumulator. The value *operand* is not affected. ADD and ADDC function identically except that ADDC adds the value of operand as well as the value of the Carry flag whereas ADD does not add the Carry flag to the result.

The **Carry bit (C)** is set if there is a carry-out of bit 7. In other words, if the unsigned summed value of the Accumulator, *operand* and (in the case of ADDC) the Carry flag exceeds 255 Carry is set. Otherwise, the Carry bit is cleared.

The **Auxillary Carry (AC)** bit is set if there is a carry-out of bit 3. In other words, if the unsigned summed value of the low nibble of the Accumulator, *operand* and (in the case of ADDC) the Carry flag exceeds 15 the Auxillary Carry flag is set. Otherwise, the Auxillary Carry flag is cleared.

The **Overflow (OV)** bit is set if there is a carry-out of bit 6 or out of bit 7, but not both. In other words, if the addition of the Accumulator, *operand* and (in the case of ADDC) the Carry flag treated as signed values results in a value that is out of the range of a signed byte (-128 through +127) the Overflow flag is set. Otherwise, the Overflow flag is cleared.

See Also: SUBB, DA, INC, DEC

8051 Instruction Set: AJMP

Operation: AJMP
Function: Absolute Jump Within 2K Block
Syntax: AJMP *code address*

Instructions	OpCode	Bytes	Flags
AJMP *page0*	0x01	2	None
AJMP *page1*	0x21	2	None
AJMP *page2*	0x41	2	None
AJMP *page3*	0x61	2	None
AJMP *page4*	0x81	2	None
AJMP *page5*	0xA1	2	None
AJMP *page6*	0xC1	2	None
AJMP *page7*	0xE1	2	None

Description: AJMP unconditionally jumps to the indicated *code address*. The new value for the Program Counter is calculated by replacing the least-significant-byte of the Program Counter with the second byte of the AJMP instruction, and replacing bits 0-2 of the most-significant-byte of the Program Counter with 3 bits that indicate the page *of the byte following the AJMP instruction*. Bits 3-7 of the most-significant-byte of the Program Counter remain unchaged.

Since only 11 bits of the Program Counter are affected by AJMP, jumps may only be made to code located within the same 2k block as the first byte that follows AJMP.

See Also: LJMP, SJMP

8051 Instruction Set: ANL

Operation: ANL
Function: Bitwise AND
Syntax: ANL *operand1*, *operand2*

Instructions	OpCode	Bytes	Flags
ANL *iram addr*,A	0x52	2	None
ANL *iram addr*,#*data*	0x53	3	None

ANL A,*#data*	0x54	2	None
ANL A,*iram addr*	0x55	2	None
ANL A,@R0	0x56	1	None
ANL A,@R1	0x57	1	None
ANL A,R0	0x58	1	None
ANL A,R1	0x59	1	None
ANL A,R2	0x5A	1	None
ANL A,R3	0x5B	1	None
ANL A,R4	0x5C	1	None
ANL A,R5	0x5D	1	None
ANL A,R6	0x5E	1	None
ANL A,R7	0x5F	1	None
ANL C,*bit addr*	0x82	2	C
ANL C,/*bit addr*	0xB0	2	C

Description: ANL does a bitwise "AND" operation between *operand1* and *operand2*, leaving the resulting value in *operand1*. The value of operand2 is not affected. A logical "AND" compares the bits of each operand and sets the corresponding bit in the resulting byte only if the bit was set in both of the original operands, otherwise the resulting bit is cleared.

See Also: ORL, XRL

8051 Instruction Set: CJNE

Operation: CJNE
Function: Compare and Jump If Not Equal
Syntax: CJNE *operand1*,*operand2*,*reladdr*

Instructions	OpCode	Bytes	Flags
CJNE A,*#data*,*reladdr*	0xB4	3	C
CJNE A,*iram addr*,*reladdr*	0xB5	3	C
CJNE @R0,*#data*,*reladdr*	0xB6	3	C
CJNE @R1,*#data*,*reladdr*	0xB7	3	C
CJNE R0,*#data*,*reladdr*	0xB8	3	C

CJNE R1,*#data,reladdr*	0xB9	3	C
CJNE R2,*#data,reladdr*	0xBA	3	C
CJNE R3,*#data,reladdr*	0xBB	3	C
CJNE R4,*#data,reladdr*	0xBC	3	C
CJNE R5,*#data,reladdr*	0xBD	3	C
CJNE R6,*#data,reladdr*	0xBE	3	C
CJNE R7,*#data,reladdr*	0xBF	3	C

Description: CJNE compares the value of *operand1* and *operand2* and branches to the indicated relative address if *operand1* and *operand2* are not equal. If the two operands are equal program flow continues with the instruction following the CJNE instruction.

The **Carry bit (C)** is set if *operand1* is less than *operand2*, otherwise it is cleared.

See Also: DJNZ

8051 Instruction Set: CLR

Operation: CLR
Function: Clear Register
Syntax: CLR *register*

Instructions	OpCode	Bytes	Flags
CLR *bit addr*	0xC2	2	None
CLR C	0xC3	1	C
CLR A	0xE4	1	None

Description: CLR clears (sets to 0) all the bit(s) of the indicated register. If the register is a bit (including the carry bit), only the specified bit is affected. Clearing the Accumulator sets the Accumulator's value to 0.

See Also: SETB

8051 Instruction Set: CPL

Operation: CPL
Function: Complement Register

Syntax: CPL *operand*

Instructions	OpCode	Bytes	Flags
CPL A	0xF4	1	None
CPL C	0xB3	1	C
CPL *bit addr*	0xB2	2	None

Description: CPL complements *operand*, leaving the result in *operand*. If *operand* is a single bit then the state of the bit will be reversed. If *operand* is the Accumulator then all the bits in the Accumulator will be reversed. This can be thought of as "Accumulator Logical Exclusive OR 255" or as "255-Accumulator." If the *operand* refers to a bit of an output Port, the value that will be complemented is based on the last value written to that bit, not the last value read from it.

See Also: CLR, SETB

8051 Instruction Set: DA

Operation: DA
Function: Decimal Adjust Accumulator
Syntax: DA A

Instructions	OpCode	Bytes	Flags
DA	0xD4	1	C

Description: DA adjusts the contents of the Accumulator to correspond to a BCD (Binary Coded Decimal) number after two BCD numbers have been added by the ADD or ADDC instruction. If the carry bit is set or if the value of bits 0-3 exceed 9, 0x06 is added to the accumulator. If the carry bit was set when the instruction began, or if 0x06 was added to the accumulator in the first step, 0x60 is added to the accumulator.

The **Carry bit (C)** is set if the resulting value is greater than 0x99, otherwise it is cleared.

See Also: ADD, ADDC

8051 Instruction Set: DEC

Operation: DEC
Function: Decrement Register
Syntax: DEC *register*

Instructions	OpCode	Bytes	Flags
DEC A	0x14	1	None
DEC *iram addr*	0x15	2	None
DEC @R0	0x16	1	None
DEC @R1	0x17	1	None
DEC R0	0x18	1	None
DEC R1	0x19	1	None
DEC R2	0x1A	1	None
DEC R3	0x1B	1	None
DEC R4	0x1C	1	None
DEC R5	0x1D	1	None
DEC R6	0x1E	1	None
DEC R7	0x1F	1	None

Description: DEC decrements the value of *register* by 1. If the initial value of *register* is 0, decrementing the value will cause it to reset to 255 (0xFF Hex). Note: The Carry Flag is NOT set when the value "rolls over" from 0 to 255.

See Also: INC, SUBB

8051 Instruction Set: DIV

Operation: DIV
Function: Divide Accumulator by B
Syntax: DIV AB

Instructions	OpCode	Bytes	Flags
DIV AB	0x84	1	C, OV

Description: Divides the unsigned value of the Accumulator by the unsigned value of the "B" register. The resulting quotient is placed in the Accumulator and the remainder is placed in the "B" register.

The **Carry flag (C)** is always cleared.

The **Overflow flag (OV)** is set if division by 0 was attempted, otherwise it is cleared.

See Also: MUL AB

8051 Instruction Set: DJNZ

Operation: DJNZ
Function: Decrement and Jump if Not Zero
Syntax: DJNZ *register,reladdr*

Instructions	OpCode	Bytes	Flags
DJNZ *iram addr,reladdr*	0xD5	3	None
DJNZ R0,*reladdr*	0xD8	2	None
DJNZ R1,*reladdr*	0xD9	2	None
DJNZ R2,*reladdr*	0xDA	2	None
DJNZ R3,*reladdr*	0xDB	2	None
DJNZ R4,*reladdr*	0xDC	2	None
DJNZ R5,*reladdr*	0xDD	2	None
DJNZ R6,*reladdr*	0xDE	2	None
DJNZ R7,*reladdr*	0xDF	2	None

Description: DJNZ decrements the value of *register* by 1. If the initial value of *register* is 0, decrementing the value will cause it to reset to 255 (0xFF Hex). If the new value of *register* is not 0 the program will branch to the address indicated by *relative addr*. If the new value of *register* is 0 program flow continues with the instruction following the DJNZ instruction.

See Also: DEC, JZ, JNZ

8051 Instruction Set: INC

Operation: INC
Function: Increment Register
Syntax: INC *register*

Instructions	OpCode	Bytes	Flags
INC A	0x04	1	None
INC *iram addr*	0x05	2	None
INC @R0	0x06	1	None
INC @R1	0x07	1	None

INC R0	0x08	1	None
INC R1	0x09	1	None
INC R2	0x0A	1	None
INC R3	0x0B	1	None
INC R4	0x0C	1	None
INC R5	0x0D	1	None
INC R6	0x0E	1	None
INC R7	0x0F	1	None
INC DPTR	0xA3	1	None

Description: INC increments the value of *register* by 1. If the initial value of *register* is 255 (0xFF Hex), incrementing the value will cause it to reset to 0. Note: The Carry Flag is NOT set when the value "rolls over" from 255 to 0.

In the case of "INC DPTR", the value two-byte unsigned integer value of DPTR is incremented. If the initial value of DPTR is 65535 (0xFFFF Hex), incrementing the value will cause it to reset to 0. Again, the Carry Flag is NOT set when the value of DPTR "rolls over" from 65535 to 0.

See Also: ADD, ADDC, DEC

8051 Instruction Set: JB

Operation: JB
Function: Jump if Bit Set
Syntax: JB *bit addr*, *reladdr*

Instructions	OpCode	Bytes	Flags
JB *bit addr*,*reladdr*	0x20	3	None

Description: JB branches to the address indicated by *reladdr* if the bit indicated by *bit addr* is set. If the bit is not set program execution continues with the instruction following the JB instruction.

See Also: JBC, JNB

8051 Instruction Set: JBC

Operation: JBC
Function: Jump if Bit Set and Clear Bit
Syntax: JB *bit addr, reladdr*

Instructions	OpCode	Bytes	Flags
JBC *bit addr,reladdr*	0x10	3	None

Description: JBC will branch to the address indicated by *reladdr* if the bit indicated by *bit addr* is set. Before branching to *reladdr* the instruction will clear the indicated bit. If the bit is not set program execution continues with the instruction following the JBC instruction.

See Also: JB, JNB

8051 Instruction Set: JC

Operation: JC
Function: Jump if Carry Set
Syntax: JC *reladdr*

Instructions	OpCode	Bytes	Flags
JC *reladdr*	0x40	2	None

Description: JC will branch to the address indicated by *reladdr* if the Carry Bit is set. If the Carry Bit is not set program execution continues with the instruction following the JC instruction.

See Also: JNC

8051 Instruction Set: JMP

Operation: JMP
Function: Jump to Data Pointer + Accumulator
Syntax: JMP @A+DPTR

Instructions	OpCode	Bytes	Flags
JMP @A+DPTR	0x73	1	None

Description: JMP jumps unconditionally to the address represented by the sum of the value of DPTR and the value of the Accumulator.

See Also: LJMP, AJMP, SJMP

8051 Instruction Set: JNB

Operation: JNB
Function: Jump if Bit Not Set
Syntax: JNB *bit addr,reladdr*

Instructions	OpCode	Bytes	Flags
JNB *bit addr,reladdr*	0x30	3	None

Description: JNB will branch to the address indicated by *reladdress* if the indicated bit is not set. If the bit is set program execution continues with the instruction following the JNB instruction.

See Also: JB, JBC

8051 Instruction Set: JNC

Operation: JNC
Function: Jump if Carry Not Set
Syntax: JNC *reladdr*

Instructions	OpCode	Bytes	Flags
JNC *reladdr*	0x50	2	None

Description: JNC branches to the address indicated by *reladdr* if the carry bit is not set. If the carry bit is set program execution continues with the instruction following the JNB instruction.

See Also: JC

8051 Instruction Set: JNZ

Operation: JNZ
Function: Jump if Accumulator Not Zero
Syntax: JNZ *reladdr*

Instructions	OpCode	Bytes	Flags
JNZ *reladdr*	0x70	2	None

Description: JNZ will branch to the address indicated by *reladdr* if the Accumulator contains any value except 0. If the value of the Accumulator is zero program execution continues with the instruction following the JNZ instruction.

See Also: JZ

8051 Instruction Set: JZ

Operation: JZ
Function: Jump if Accumulator Zero
Syntax: JNZ *reladdr*

Instructions	OpCode	Bytes	Flags
JZ *reladdr*	0x60	2	None

Description: JZ branches to the address indicated by *reladdr* if the Accumulator contains the value 0. If the value of the Accumulator is non-zero program execution continues with the instruction following the JNZ instruction.

See Also: JNZ

8051 Instruction Set: LCALL

Operation: LCALL
Function: Long Call
Syntax: LCALL *code addr*

Instructions	OpCode	Bytes	Flags
LCALL *code addr*	0x12	3	None

Description: LCALL calls a program subroutine. LCALL increments the program counter by 3 (to point to the instruction following LCALL) and pushes that value onto the stack (low byte first, high byte second). The Program Counter is then set to the 16-bit value which follows the LCALL opcode, causing program execution to continue at that address.

See Also: ACALL, RET

8051 Instruction Set: LJMP

Operation: LJMP

Function: Long Jump
Syntax: LJMP *code addr*

Instructions	OpCode	Bytes	Flags
LJMP *code addr*	0x02	3	None

Description: LJMP jumps unconditionally to the specified *code addr*.

See Also: AJMP, SJMP, JMP

8051 Instruction Set: MOV

Operation: MOV
Function: Move Memory
Syntax: MOV *operand1*,*operand2*

Instructions	OpCode	Bytes	Flags
MOV @R0,*#data*	0x76	2	None
MOV @R1,*#data*	0x77	2	None
MOV @R0,A	0xF6	1	None
MOV @R1,A	0xF7	1	None
MOV @R0,*iram addr*	0xA6	2	None
MOV @R1,*iram addr*	0xA7	2	None
MOV A,*#data*	0x74	2	None
MOV A,@R0	0xE6	1	None
MOV A,@R1	0xE7	1	None
MOV A,R0	0xE8	1	None
MOV A,R1	0xE9	1	None
MOV A,R2	0xEA	1	None
MOV A,R3	0xEB	1	None
MOV A,R4	0xEC	1	None
MOV A,R5	0xED	1	None
MOV A,R6	0xEE	1	None
MOV A,R7	0xEF	1	None

MOV A,*iram addr*	0xE5	2	None
MOV C,*bit addr*	0xA2	2	C
MOV DPTR,*#data16*	0x90	3	None
MOV R0,*#data*	0x78	2	None
MOV R1,*#data*	0x79	2	None
MOV R2,*#data*	0x7A	2	None
MOV R3,*#data*	0x7B	2	None
MOV R4,*#data*	0x7C	2	None
MOV R5,*#data*	0x7D	2	None
MOV R6,*#data*	0x7E	2	None
MOV R7,*#data*	0x7F	2	None
MOV R0,A	0xF8	1	None
MOV R1,A	0xF9	1	None
MOV R2,A	0xFA	1	None
MOV R3,A	0xFB	1	None
MOV R4,A	0xFC	1	None
MOV R5,A	0xFD	1	None
MOV R6,A	0xFE	1	None
MOV R7,A	0xFF	1	None
MOV R0,*iram addr*	0xA8	2	None
MOV R1,*iram addr*	0xA9	2	None
MOV R2,*iram addr*	0xAA	2	None
MOV R3,*iram addr*	0xAB	2	None
MOV R4,*iram addr*	0xAC	2	None
MOV R5,*iram addr*	0xAD	2	None
MOV R6,*iram addr*	0xAE	2	None
MOV R7,*iram addr*	0xAF	2	None
MOV *bit addr*,C	0x92	2	None
MOV *iram addr*,*#data*	0x75	3	None
MOV *iram addr*,@R0	0x86	2	None

MOV *iram addr*,@R1	0x87	2	None
MOV *iram addr*,R0	0x88	2	None
MOV *iram addr*,R1	0x89	2	None
MOV *iram addr*,R2	0x8A	2	None
MOV *iram addr*,R3	0x8B	2	None
MOV *iram addr*,R4	0x8C	2	None
MOV *iram addr*,R5	0x8D	2	None
MOV *iram addr*,R6	0x8E	2	None
MOV *iram addr*,R7	0x8F	2	None
MOV *iram addr*,A	0xF5	2	None
MOV *iram addr*,*iram addr*	0x85	3	None

Description: MOV copies the value of *operand2* into *operand1*. The value of *operand2* is not affected. Both *operand1* and *operand2* must be in Internal RAM. No flags are affected unless the instruction is moving the value of a bit into the carry bit in which case the carry bit is affected or unless the instruction is moving a value into the PSW register (which contains all the program flags).

** Note: In the case of "MOV iram addr,iram addr", the operand bytes of the instruction are stored in reverse order. That is, the instruction consisting of the bytes 0x85, 0x20, 0x50 means "Move the contents of Internal RAM location 0x20 to Internal RAM location 0x50" whereas the opposite would be generally presumed.

See Also: MOVC, MOVX, XCH, XCHD, PUSH, POP

8051 Instruction Set: MOVC

Operation: MOVC
Function: Move Code Byte to Accumulator
Syntax: MOVC A,@A+*register*

Instructions	OpCode	Bytes	Flags
MOVC A,@A+DPTR	0x93	1	None
MOVC A,@A+PC	0x83	1	None

Description: MOVC moves a byte from Code Memory into the Accumulator. The Code Memory address from which the byte will be moved is calculated by summing the value of

the Accumulator with either DPTR or the Program Counter (PC). In the case of the Program Counter, PC is first incremented by 1 before being summed with the Accumulator.

See Also: MOV, MOVX

8051 Instruction Set: MOVX

Operation: MOVX
Function: Move Data To/From External Memory (XRAM)
Syntax: MOVX *operand1*,*operand2*

Instructions	OpCode	Bytes	Flags
MOVX @DPTR,A	0xF0	1	None
MOVX @R0,A	0xF2	1	None
MOVX @R1,A	0xF3	1	None
MOVX A,@DPTR	0xE0	1	None
MOVX A,@R0	0xE2	1	None
MOVX A,@R1	0xE3	1	None

Description: MOVX moves a byte to or from External Memory into or from the Accumulator.

If *operand1* is @DPTR, the Accumulator is moved to the 16-bit External Memory address indicated by DPTR. This instruction uses both P0 (port 0) and P2 (port 2) to output the 16-bit address and data. If *operand2* is DPTR then the byte is moved from External Memory into the Accumulator.

If *operand1* is @R0 or @R1, the Accumulator is moved to the 8-bit External Memory address indicated by the specified Register. This instruction uses only P0 (port 0) to output the 8-bit address and data. P2 (port 2) is not affected. If *operand2* is @R0 or @R1 then the byte is moved from External Memory into the Accumulator.

See Also: MOV, MOVC

8051 Instruction Set: MUL

Operation: MUL
Function: Multiply Accumulator by B
Syntax: MUL AB

Instructions	OpCode	Bytes	Flags
MUL AB	0xA4	1	C, OV

Description: Multiples the unsigned value of the Accumulator by the unsigned value of the "B" register. The least significant byte of the result is placed in the Accumulator and the most-significant-byte is placed in the "B" register.

The **Carry Flag (C)** is always cleared.

The **Overflow Flag (OV)** is set if the result is greater than 255 (if the most-significant byte is not zero), otherwise it is cleared.

See Also: DIV

8051 Instruction Set: NOP

Operation: NOP
Function: None, waste time
Syntax: No Operation

Instructions	OpCode	Bytes	Flags
NOP	0x00	1	None

Description: NOP, as it's name suggests, causes No Operation to take place for one machine cycle. NOP is generally used only for timing purposes. Absolutely no flags or registers are affected.

8051 Instruction Set: ORL

Operation: ORL
Function: Bitwise OR
Syntax: ORL *operand1,operand2*

Instructions	OpCode	Bytes	Flags
ORL *iram addr*,A	0x42	2	None
ORL *iram addr,#data*	0x43	3	None
ORL A,*#data*	0x44	2	None
ORL A,*iram addr*	0x45	2	None

ORL A,@R0	0x46	1	None
ORL A,@R1	0x47	1	None
ORL A,R0	0x48	1	None
ORL A,R1	0x49	1	None
ORL A,R2	0x4A	1	None
ORL A,R3	0x4B	1	None
ORL A,R4	0x4C	1	None
ORL A,R5	0x4D	1	None
ORL A,R6	0x4E	1	None
ORL A,R7	0x4F	1	None
ORL C,*bit addr*	0x72	2	C
ORL C,/*bit addr*	0xA0	2	C

Description: ORL does a bitwise "OR" operation between *operand1* and *operand2*, leaving the resulting value in *operand1*. The value of *operand2* is not affected. A logical "OR" compares the bits of each operand and sets the corresponding bit in the resulting byte if the bit was set in either of the original operands, otherwise the resulting bit is cleared.

See Also: ANL, XRL

8051 Instruction Set: POP

Operation: POP
Function: Pop Value From Stack
Syntax: POP

Instructions	OpCode	Bytes	Flags
POP *iram addr*	0xD0	2	None

Description: POP "pops" the last value placed on the stack into the *iram addr* specified. In other words, POP will load *iram addr* with the value of the Internal RAM address pointed to by the current Stack Pointer. The stack pointer is then decremented by 1.

See Also: PUSH

8051 Instruction Set: PUSH

Operation: PUSH
Function: Push Value Onto Stack
Syntax: PUSH

Instructions	OpCode	Bytes	Flags
PUSH *iram addr*	0xC0	2	None

Description: PUSH "pushes" the value of the specified *iram addr* onto the stack. PUSH first increments the value of the Stack Pointer by 1, then takes the value stored in *iram addr* and stores it in Internal RAM at the location pointed to by the incremented Stack Pointer.

See Also: POP

8051 Instruction Set: RET

Operation: RET
Function: Return From Subroutine
Syntax: RET

Instructions	OpCode	Bytes	Flags
RET	0x22	1	None

Description: RET is used to return from a subroutine previously called by LCALL or ACALL. Program execution continues at the address that is calculated by popping the topmost 2 bytes off the stack. The most-significant-byte is popped off the stack first, followed by the least-significant-byte.

See Also: LCALL, ACALL, RETI

8051 Instruction Set: RETI

Operation: RETI
Function: Return From Interrupt
Syntax: RETI

Instructions	OpCode	Bytes	Flags
RETI	0x32	1	None

Description: RETI is used to return from an interrupt service routine. RETI first enables interrupts of equal and lower priorities to the interrupt that is terminating. Program execution

continues at the address that is calculated by popping the topmost 2 bytes off the stack. The most-significant-byte is popped off the stack first, followed by the least-significant-byte.

RETI functions identically to RET if it is executed outside of an interrupt service routine.

See Also: RET

8051 Instruction Set: RL

Operation: RL
Function: Rotate Accumulator Left
Syntax: RL A

Instructions	OpCode	Bytes	Flags
RL A	0x23	1	C

Description: Shifts the bits of the Accumulator to the left. The left-most bit (bit 7) of the Accumulator is loaded into bit 0.

See Also: RLC, RR, RRC

8051 Instruction Set: RLC

Operation: RLC
Function: Rotate Accumulator Left Through Carry
Syntax: RLC A

Instructions	OpCode	Bytes	Flags
RLC A	0x33	1	C

Description: Shifts the bits of the Accumulator to the left. The left-most bit (bit 7) of the Accumulator is loaded into the Carry Flag, and the original Carry Flag is loaded into bit 0 of the Accumulator. This function can be used to quickly multiply a byte by 2.

See Also: RL, RR, RRC

8051 Instruction Set: RR

Operation: RR

Function: Rotate Accumulator Right
Syntax: RR A

Instructions	OpCode	Bytes	Flags
RR A	0x03	1	None

Description: Shifts the bits of the Accumulator to the right. The right-most bit (bit 0) of the Accumulator is loaded into bit 7.

See Also: RL, RLC, RRC

8051 Instruction Set: RRC

Operation: RRC
Function: Rotate Accumulator Right Through Carry
Syntax: RRC A

Instructions	OpCode	Bytes	Flags
RRC A	0x13	1	C

Description: Shifts the bits of the Accumulator to the right. The right-most bit (bit 0) of the Accumulator is loaded into the Carry Flag, and the original Carry Flag is loaded into bit 7. This function can be used to quickly divide a byte by 2.

See Also: RL, RLC, RR

8051 Instruction Set: SETB

Operation: SETB
Function: Set Bit
Syntax: SETB *bit addr*

Instructions	OpCode	Bytes	Flags
SETB C	0xD3	1	C
SETB *bit addr*	0xD2	2	None

Description: Sets the specified bit.

See Also: CLR

8051 Instruction Set: SJMP

Operation: SJMP
Function: Short Jump
Syntax: SJMP *reladdr*

Instructions	OpCode	Bytes	Flags
SJMP *reladdr*	0x80	2	None

Description: SJMP jumps unconditionally to the address specified *reladdr*. *Reladdr* must be within -128 or +127 bytes of the instruction that follows the SJMP instruction.

See Also: LJMP, AJMP

8051 Instruction Set: SUBB

Operation: SUBB
Function: Subtract from Accumulator With Borrow
Syntax: SUBB A,*operand*

Instructions	OpCode	Bytes	Flags
SUBB A,*#data*	0x94	2	C, AC, OV
SUBB A,*iram addr*	0x95	2	C, AC, OV
SUBB A,@R0	0x96	1	C, AC, OV
SUBB A,@R1	0x97	1	C, AC, OV
SUBB A,R0	0x98	1	C, AC, OV
SUBB A,R1	0x99	1	C, AC, OV
SUBB A,R2	0x9A	1	C, AC, OV
SUBB A,R3	0x9B	1	C, AC, OV
SUBB A,R4	0x9C	1	C, AC, OV
SUBB A,R5	0x9D	1	C, AC, OV
SUBB A,R6	0x9E	1	C, AC, OV
SUBB A,R7	0x9F	1	C, AC, OV

Description: SUBB subtract the value of *operand* from the value of the Accumulator, leaving the resulting value in the Accumulator. The value *operand* is not affected.

The **Carry Bit (C)** is set if a borrow was required for bit 7, otherwise it is cleared. In other words, if the unsigned value being subtracted is greater than the Accumulator the Carry Flag is set.

The **Auxillary Carry (AC)** bit is set if a borrow was required for bit 3, otherwise it is cleared. In other words, the bit is set if the low nibble of the value being subtracted was greater than the low nibble of the Accumulator.

The **Overflow (OV)** bit is set if a borrow was required for bit 6 or for bit 7, but not both. In other words, the subtraction of two signed bytes resulted in a value outside the range of a signed byte (-128 to 127). Otherwise it is cleared.

See Also: ADD, ADDC, DEC

8051 Instruction Set: SWAP

Operation: SWAP
Function: Swap Accumulator Nibbles
Syntax: SWAP A

Instructions	OpCode	Bytes	Flags
SWAP A	0xC4	1	None

Description: SWAP swaps bits 0-3 of the Accumulator with bits 4-7 of the Accumulator. This instruction is identical to executing "RR A" or "RL A" four times.

See Also: RL, RLC, RR, RRC

8051 Instruction Set: Undefined Instruction

Operation: Undefined Instruction
Function: Undefined
Syntax: ???

Instructions	OpCode	Bytes	Flags
???	0xA5	1	C

Description: The "Undefined" instruction is, as the name suggests, not a documented instruction. The 8051 supports 255 instructions and OpCode 0xA5 is the single OpCode that is not used by any documented function. Since it is not documented nor defined it is not recommended that it be executed. However, based on my research, executing this undefined instruction takes 1 machine cycle and appears to have no effect on the system except that the Carry Bit always seems to be set.

Note: We received input from an 8052.com user that the undefined instruction really has a format of *Undefined bit1,bit2* and effectively copies the value of bit2 to bit1. In this case, it would be a three-byte instruction. We haven't had an opportunity to verify or disprove this report, so we present it to the world as "additional information."

Note: It has been reported that Philips 8051 model P89C669 uses instruction prefix 0xA5 to let the user access a different (extended) SFR area.

8051 Instruction Set: XCH

Operation: XCH
Function: Exchange Bytes
Syntax: XCH A,*register*

Instructions	OpCode	Bytes	Flags
XCH A,@R0	0xC6	1	None
XCH A,@R1	0xC7	1	None
XCH A,R0	0xC8	1	None
XCH A,R1	0xC9	1	None
XCH A,R2	0xCA	1	None
XCH A,R3	0xCB	1	None
XCH A,R4	0xCC	1	None
XCH A,R5	0xCD	1	None
XCH A,R6	0xCE	1	None
XCH A,R7	0xCF	1	None
XCH A,*iram addr*	0xC5	2	None

Description: Exchanges the value of the Accumulator with the value contained in *register*.

See Also: MOV

8051 Instruction Set: XCHD

Operation: XCHD
Function: Exchange Digit
Syntax: XCHD A,*[@R0/@R1]*

Instructions	OpCode	Bytes	Flags
XCHD A,@R0	0xD6	1	None
XCHD A,@R1	0xD7	1	None

Description: Exchanges bits 0-3 of the Accumulator with bits 0-3 of the Internal RAM address pointed to indirectly by R0 or R1. Bits 4-7 of each register are unaffected.

See Also: DA

8051 Instruction Set: XRL

Operation: XRL
Function: Bitwise Exclusive OR
Syntax: XRL *operand1,operand2*

Instructions	OpCode	Bytes	Flags
XRL *iram addr*,A	0x62	2	None
XRL *iram addr,#data*	0x63	3	None
XRL A,*#data*	0x64	2	None
XRL A,*iram addr*	0x65	2	None
XRL A,@R0	0x66	1	None
XRL A,@R1	0x67	1	None
XRL A,R0	0x68	1	None
XRL A,R1	0x69	1	None
XRL A,R2	0x6A	1	None
XRL A,R3	0x6B	1	None
XRL A,R4	0x6C	1	None
XRL A,R5	0x6D	1	None
XRL A,R6	0x6E	1	None
XRL A,R7	0x6F	1	None

Description: XRL does a bitwise "EXCLUSIVE OR" operation between *operand1* and *operand2*, leaving the resulting value in *operand1*. The value of operand2 is not affected. A logical "EXCLUSIVE OR" compares the bits of each operand and sets the corresponding bit in the resulting byte if the bit was set in either (but not both) of the original operands, otherwise the bit is cleared.