Instruction set of Microcontroller 8051

The instructions of 8051 Microcontroller can be classified into five different groups. These groups are like below

- Data Transfer Group
- Arithmetic Group
- Logical Group
- Program Branch Group
- Bit Processing Group

1. Data Transfer Instructions

Operation: MOV
Syntax: MOV destination, source

Description: MOV copies the value of source into destination. The value of source is not affected. Both destination and source 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).

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

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.

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

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.

Operation: SWAP

Function: Swap Accumulator Nibbles

Syntax: SWAP A

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.

Operation: XCH

Function: Exchange Bytes

Syntax: XCH A,*register*

Description: Exchanges the value of the Accumulator with the value contained in register.
Ex: XCH A, R1

Operation: PUSH

Function: Push Value onto Stack

Syntax: PUSH

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.

Operation: POP

Function: Pop Value from Stack

Syntax: POP

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.

2. Arithmetic Instructions

Operation: ADD, ADDC

Function: Add Accumulator, Add Accumulator with Carry

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.

Operation: SUBB

Function: Subtract from Accumulator With Borrow

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.

Operation: MUL

Function: Multiply Accumulator by B

Syntax: MUL AB

Description: Multiples the unsigned value of the Accumulator by the unsigned value of the “B” registers. 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.

Operation: DIV

Function: Divide Accumulator by B

Syntax: DIV AB

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.

Operation: INC

Function: Increment Register

Syntax: INC *register*

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.

Operation: DEC

Function: Decrement Register

Syntax: DEC *register*

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.

3. Logical Instructions

Operation: ORL

Function: Bitwise OR

Syntax: ORL *operand1*,*operand2*

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.

Operation: ANL

Function: Bitwise AND

Syntax: ANL *operand1*, *operand2*

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.

Operation: XRL

Function: Bitwise Exclusive OR

Syntax: XRL *operand1*,*operand2*

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.

Operation: CPL

Function: Complement Register

Syntax: CPL *operand*

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.

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

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.

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

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

Operation: RR
Function: Rotate Accumulator Right
Syntax: RR A

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

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

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.

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

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.

4. Branching Instructions

Operation: JMP

Function: Jump to Data Pointer + Accumulator

Syntax: JMP @A+DPTR

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

Operation: JC

Function: Jump if Carry Set

Syntax: JC *reladdr*

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.

Operation: JNC

Function: Jump if Carry Not Set

Syntax: JNC *reladdr*

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.

Operation: DJNZ (Loop Instructions)

Function: Decrement and Jump if Not Zero

Syntax: DJNZ *register,reladdr*

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.

Operation: CJNE

Function: Compare and Jump If Not Equal

Syntax: CJNE *operand1,operand2,reladdr*

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.

Operation: JZ

Function: Jump if Accumulator Zero

Syntax: JNZ *reladdr*

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.

Operation: JNZ

Function: Jump if Accumulator Not Zero

Syntax: JNZ *reladdr*

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.

Operation: SJMP

Function: Short Jump

Syntax: SJMP *reladdr*

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.

Operation: LJMP

Function: Long Jump

Syntax: LJMP *code addr*

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

Operation: AJMP

Function: Absolute Jump Within 2K Block

Syntax: AJMP *code address*

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 unchanged.

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.

Operation: LCALL **(CALL Instructions)**

Function: Long Call

Syntax: LCALL *code addr*

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.

Operation: ACALL

Function: Absolute Call Within 2K Block

Syntax: ACALL *code address*

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 unchanged.

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.

Operation: RET

Function: Return From Subroutine

Syntax: RET

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.

Operation: RETI

Function: Return from Interrupt

Syntax: RETI

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.

5. Bit-Wise Instructions

Operation: JB

Function: Jump if Bit Set

Syntax: JB *bit addr, reladdr*

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.

Operation: JNB

Function: Jump if Bit Not Set

Syntax: JNB *bit addr,reladdr*

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.

Operation: JBC

Function: Jump if Bit Set and Clear Bit

Syntax: JB *bit addr, reladdr*

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.