e-PG Pathshala

Subject : Computer Science

Paper: Embedded System
Module: Microcontrollers and Embedded Processors
Module No: CS/ES/7

Quadrant 1 ï e-text

In this lecture, Instruction set of 8051 microcontroller will be discussed in detail. Arithmetic and logical instructions, and control transfer instructions will be discussed with examples.

1.Instruction set

The process of writing program for the microcontroller mainly consists of giving instructions (commands) in the specific order in which they should be executed in order to carry out a specific task. As digital circuit cannot ñunderstandò what for example an instruction ñif the push button is pressed- turn the light onò means, then a certain number of simpler and precisely defined orders that decoder can recognize must be used. All commands are known as INSTRUCTION SET. All microcontrollers compatible with the 8051 have a total of 255 instructions.

In these 255 instructions, many instructions are considered to be ñdifferentò, even though they perform the same operation, so there are only 111 truly different commands. For example: ADD A,R0, ADD A,R1, ... ADD A,R7 are instructions that perform the same operation (addition of the accumulator and register). Since there are 8 such registers, each instruction is counted separately. Taking into account that all instructions perform only 53 operations (addition, subtraction, copy etc.) and most of them are rarely used in practice, there are actually 20-30 abbreviations to be learned, which is acceptable.

1.1 Types of instructions

Depending on the operation that they perform, the instructions are divided into several groups:

- Arithmetic Instructions
- Branch Instructions
- Data Transfer Instructions
- Logic Instructions and
- Bit-oriented Instructions.

The first part of each instruction, called MNEMONIC refers to the operation an instruction performs (copy, addition, logic operation etc.). Mnemonics are abbreviations of the name of operation being executed. For example:

ADD A, R0 - ADD mnemonic (A, R0- operands registers)

1.1.1 Arithmetic Instructions

Arithmetic instructions perform addition, subtraction, multiplication, division, increment and decrement operations. After execution, the result is stored in the first operand. For example: *ADD* A,R1 - The result of addition (A+R1) will be stored in the accumulator.

Table .1 Execution Table for Arithmetic instructions

Mnemonic	Operation	Addressing Modes				Execution Time in X1 Mode @12 MHz (µs)
		Dir	Ind	Reg	Imm	
ADD A, <byte>	A = A + <byte>	X	X	X	X	
ADDC A, <byte>	A = A + <byte> + C	X	X	X	X	1
SUBB A, <byte>	A = A – <byte> – C	X	X	X	X	1
INC A	A = A + 1	Accumulator only				1
INC <byte>	<byte> = <byte> + 1	X	X	X		1
INC DPTR	DPTR = DPTR + 1	Data Pointer only				2
DEC A	A = A – 1	Accumulator only				1
DEC <byte>	<byte> = <byte> – 1	X	X	X		1
MUL AB	B:A = B × A	ACC and B only				4
DIV AB	A = Int [A/B] B = Mod [A/B]	ACC and B only				4
DA A	Decimal Adjust	Accumulator only				1

From Table.1 it is evident that always one of the operand is stored in accumulator. Addition is done through ADD and ADDC instruction. Addition with carry is needed when doing multiple addition. Subtraction is always done with borrow using SUBB instruction. Increment and decrement by one bit can be done with INC and DEC instructions. The DA A instruction is the decimal adjust used for BCD addition.

The MUL AB instruction multiplies the Accumulator by the data in the B register and puts the 16-bit product into the concatenated B and Accumulator registers. The DIV AB instruction divides the Accumulator by the data in the B register and leaves the 8-bit quotient in the Accumulator, and the 8-bit remainder in the B register.

Examples for addition and subtraction:

ADD A,<src-byte>
Assume A=09h and R0=07h
ADD A,R0 ; (A) Ŷ (A) + (R0), A=10h
ADDC A,Rn ; (A) Ŷ (A) + (C) + (Rn)

In the ADDC instruction of previous example if the carry bit is 1 then 10h+1 is 11h and that is stored in Accumulator.

SUBB A,<src-byte>
Assume A=09h, R0=07h, carry flag=1
SUBB A,R0 ; (A) Ŷ (A) - (C) - (R0) , A=1

Examples for multiplication and division :

Assume A=09h,B=02h

Å MUL AB ; (A)9 Ŷ (A) X (B)
(B)2, A=12h,B=00h
Multiplying A and B gives 18 in decimal and in hexadecimal it is 12h. Here, LSB is stored in A register and MSB is stored in B register. There is no overflow in the above instruction, if there is overflow then those values are stored in B register.

Å DIV AB ; (A)9 Ŷ (A)/(B)
(B)2 , A=04h,B=01h
Division of A and B gives quotient as 4 and remainder as 1. A holds the quotient and B holds the remainder.

1.1.2. Logical Instruction

Logic instructions perform logic operations upon corresponding bits of two registers. After execution, the result is stored in the first operand. These instructions perform basic logic operations such as OR,AND,XOR ,complement and rotate operations.

Table.2 Execution Table for Logical Instruction

Mnemonic	Operation	Addressing Modes				Execution Time @ 12MHz (µs)
		Dir	Ind	Reg	Imm	
ANL A, <byte>	A = A AND <byte>	X	X	X	X	1
ANL <byte>, A	<byte> = <byte> AND A	X				1
ANL <byte>, # data	<byte> = <byte> AND # data	X				2
ORL A, <byte>	A = A OR <byte>	X	X	X	X	1
ORL <byte>, A	<byte> = <byte> OR A	X				1
ORL <byte>, # data	<byte> = <byte> OR # data	X				2
XRL A, <byte>	A = A XOR <byte>	X	X	X	X	1
XRL <byte>, A	<byte> = <byte> XOR A	X				1
XRL <byte>, # data	<byte> = <byte> XOR # data	X				2
CLR A	A = 00H	Accumulator only				1
CLP A	A = NOT A	Accumulator only				1
RL A	Rotate ACC Left 1 bit	Accumulator only				1
RLC A	Rotate Left through Carry	Accumulator only				1
RR A	Rotate ACC Right 1 bit	Accumulator only				1
RRC A	Rotate Right through Carry	Accumulator only				1
SWAP A	Swap Nibbles in A	Accumulator only				1

All logical instructions are listed in Table.2. Accumulator holds one of the operand always. ANL performs the logical AND operation. OR operation is performed through ORL instruction. XOR operation is done with XRL instruction. Reset to zero is done with CLR instruction. Complement is done with CLP. In assembly code rotation of data is also possible. Rotate left, rotate right can be done with accumulator content through RL and RR instructions. RLC and RRC will take care about rotating accumulator data with carry. SWAP will swap lower and upper nibbles of accumulator data. The execution time for all these instructions are listed in Table.2.

Example for Logical AND operation :
AND opcode will logically AND the content of the two operands. Here the operands are at registers A and R0.

Assume A= 10011101(9DH), R0=11101110 (EEH)
ANL A,R1 which gives (A) Ŷ (A) ∧ (R0) A=10001100 (8CH)

AND operation is done bitwise i.e if the first bit of A is 1 and first bit of R0 is 1 then the AND operation of A and R0 is 1. Hence the first bit is 1 which is stored as the result in Accumulator. Thus content of A register is 8CH.

Clearing or resetting the content is done by CLR opcode.
CLR A makes (A) = 0

Complement is obtained through CPL.

A=10001100
CPL A ; (A) Ŷ (Aô) Which makes A=01100010

For swapping lower and upper nibble SWAP opcode is used, if A=10001100

SWAP A ; makes A=11011001=D9H, the LSB and MSB bits are swapped.

Instruction set provides rotation of register content also (Figure 3 & 4).

Examples for rotation:

Rotate Accumulator Left - RL A

The 8 bits of the accumulator are rotated left by one bit, and Bit D7 exits from the MSB and enters into LSB, D0
if A = 1001110
RL A makes A= 00111011(3EH),

Figure 3: Rotating left Accumulator

RLC A makes rotation include carry also.

Figure 4 : Rotating left accumulator with carry

Boolean Operations - Bit-oriented Instructions :

Similar to logic instructions, bit-oriented instructions perform logic operations (Table.3). The difference is that these are performed upon single bits. All bit accesses are by direct addressing. Bit addresses 00H through 7FH are in the Lower 128, and bit addresses 80H through FFH are in SFR (special function register)space.

Examples
ANL C,bit ; Logical-AND for bit variables
(C) Ŷ (C) ∧ (bit)
ANL C,ACC.7 ;and carry with accum.bit 7
MOV C,P1.0 ;load carry with input pin state

Table 3 Bit wise operation

Mnemonic	Operation	Execution Time @ 12MHz (µs)
ANL C,bit	C = C AND bit	2
ANL C,/bit	C = C AND (NOT bit)	2
ORL C,bit	C = C OR bit	2
ORL C,/bit	C = C OR (NOT bit)	2
MOV C,bit	C = bit	1
MOV bit,C	bit = C	2
CLR C	C = 0	1
CLR bit	bit = 0	1
SETB C	C = 1	1
SETB bit	bit = 1	1
CPL C	C = NOT C	1
CPL bit	bit = NOT bit	1
JC rel	Jump if C = 1	2
JNC rel	Jump if C = 0	2
JB bit,rel	Jump if bit = 1	2
JNB bit,rel	Jump if bit = 0	2
JBC bit,rel	Jump if bit = 1 ; CLR bit	2

SETB instruction is used for setting the value of 1. The JC rel instruction is the jump carry instruction, rel refers to the relative address. JNC is jump on No carry to a particular address. JB instruction refers to jump on bit. JNB instruction is jump if bit is equal to zero. JBC instruction is jump if bit is equal to one and clears that bit.

1.1.3 Branch Instructions

There are two kinds of branch instructions:

Unconditional jump instructions: upon their execution a jump to a new location from where the program continues execution is performed.

Conditional jump instructions: a jump to a new program location is executed only if a specified condition is met. Otherwise, the program normally proceeds with the next instruction. Table .4 gives the list of control jump instructions

In unconditional branch instruction there are three types of jump instructions, namely :

I. Short jump
II. Absolute jump
III. Long jump.

Short jump:

The SJMP instruction encodes the destination address as relative offset. The instruction is 2 bytes long, consisting of the opcode and the relative offset byte. The jump distance is limited to a range of -128 to + 127 bytes relative to the instruction following the SJMP.

Absolute jump :

The AJMP instruction encodes the destination address as an 11-bit constant. The instruction is 2 bytes long, consisting of the opcode, which itself contains 3 of the 11 address bits, followed by

another byte containing the low 8 bits of the destination address. When the instruction is executed, these 11 bits are simply substituted for the low 11 bits in the PC. The high 5 bits stay the same.
Hence the destination has to be within the same 2K block as the instruction following the AJMP.

Table 4 control jump instructions

ACALL	addr11	Absolute Subroutine Call
LCALL	addr16	Long Subroutine Call
RET		Return from Subroutine
RETI		Return from interrupt
AJMP	addr11	Absolute Jump
LJMP	addr16	Long Jump
SJMP	rel	Short Jump (relative addr)
JMP	@A+DPTR	Jump indirect relative to the DPTR
JZ	rel	Jump if Accumulator is Zero
JNZ	rel	Jump if Accumulator is Not Zero
CJNE	A,direct,rel	Compare direct byte to Acc and Jump if Not Equal
CJNE	A,#data,rel	Compare immediate to Acc and Jump if Not Equal
CJNE	R_n,#data,rel	Compare immediate to register and Jump if Not Equal
CJNE	@R,#data,rel	Compare immediate to indirect and Jump if Not Equal
DJNZ	R_n,rel	Decrement register and Jump if Not Zero
DJNZ	direct,rel	Decrement direct byte and Jump if Not Zero
NOP		No Operation

Long jump
The LJMP instruction encodes the destination address as a 16-bit constant. The instruction is 3 bytes long, consisting of the opcode and two address bytes. The destination address can be anywhere in the 64K Program Memory space.

CALL instructions are used for calling subroutines in 8051 controller.

ACALL addr11 (Absolute Call)
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 through 5, and the second byte of the instruction.

LCALL addr16 (Long call)
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.The subroutine may therefore begin anywhere in the full 64K byte program memory address space.

Conditional Jump :

DJNZ<byte>,<reladdr>(Decrement and Jump if Not Zero)
DJNZ 40H,LABEL_1; decrement content of address 40H if it is not zero, then go to LABEL_1
DJNZ R1,TOGGLE ; Decrement content of R1, if it is not zero then go to TOGGLE
CJNE <destbyte>,<src-byte>, rel (Compare and Jump if Not Equal)
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.
CJNE R7, # 60H, NOT_EQ

3. Summary

In this lecture Instruction set of 8051 is discussed. The Arithmetic and logical operations and the control transfer instructions are discussed with examples.