COURSE NAME WITH CODE: MICROCONTROLLERS(EC45) **FACULTY NAME: ROOPA M** **DATE:**	**UNIT-II**

OBJECTIVES OF THE LECTURE:

To Study on

- Immediate, direct, Indirect, Register, Indexed addressing modes
- Instruction set of 8051Data transfer, arithmetic and logical, Bit manipulation, Jump and call instructions. Assembler directives.
- Applications

REFERENCES:

1. " The 8051 Microcontroller Architecture Programming & Applications", 2e Kenneth J. Ayala
2. " The 8051 Microcontroller and Embedded Systems-using Assembly and C", Muhammad Ali Mazidi, Janice GillespieMazidi and Rllin D. McKinalay; PHI, 2006/Pearson,2006

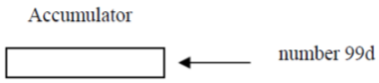
Addressing modes

There are a number of addressing modes available to the 8051 instruction set, as follows:

- Immediate Addressing
- Register Addressing
- Direct Addressing
- Indirect Addressing
- Relative Addressing
- Absolute addressing
- Indexed Addressing

❖ Immediate Addressing

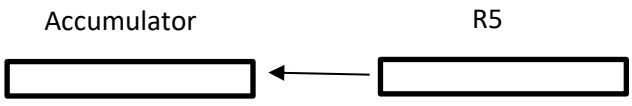
- Immediate addressing simply means that the operand (which immediately follows the instruction op. code) is the data value to be used.
- For example the instruction:
  MOV A, #99d



- Moves the value 99 into the accumulator (note this is 99 decimal since we used 99d).
- The # symbol tells the assembler that the immediate addressing mode is to be used.

❖ Register Addressing

- One of the eight general-registers, R0 to R7, can be specified as the instruction operand. The assembly language documentation refers to a register generically as Rn.
- An example instruction using register addressing is :
  ADD A, R5; Adds register R5 to A (accumulator)

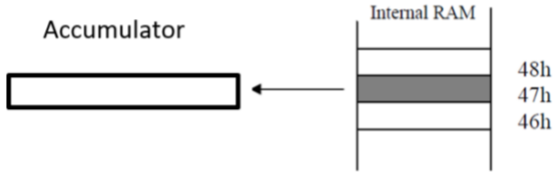


- Here the contents of R5 is added to the accumulator.
- One advantage of register addressing is that the instructions tend to be short, single byte instructions.

❖ Direct Addressing

- Direct addressing means that the data value is obtained directly from the memory location specified in the operand.
- For example consider the instruction:

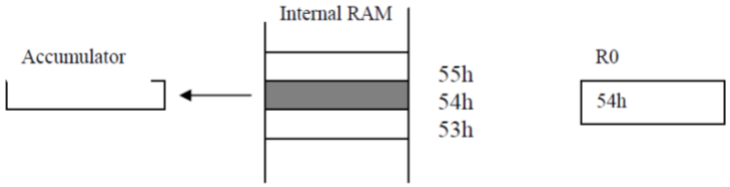
  MOV A, 47h



- The instruction reads the data from Internal RAM address 47h and stores this in the accumulator.
- Direct addressing can be used to access Internal RAM including the SFR registers.

❖ Indirect Addressing

- Indirect addressing provides a powerful addressing capability, which needs to be appreciated.
- An example instruction, which uses indirect addressing, is as follows:
  MOV A, @R0

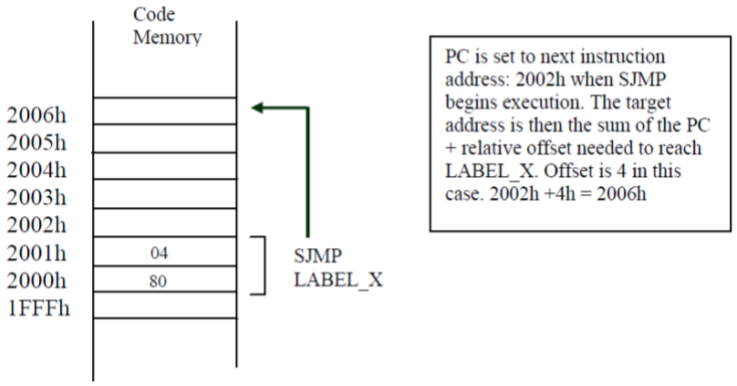


- Note @ symbol indicated that the indirect addressing mode is used.
- R0 contains a value, for example 54h, which is to be used as the address of the internal RAM location, which contains the operand data.
- Indirect addressing refers to Internal RAM only and cannot be used to refer to SFR registers.
- Note, only R0 or R1 can be used as register data pointers for indirect addressing when using MOV instructions.

❖ Relative Addressing

- This is a special addressing mode used with certain jump instructions.

- The relative address, often referred to as an offset, is an 8-bit signed number, which is automatically added to the PC to make the address of the next instruction. The 8-bit signed offset value gives an address range of + 127 to –128 locations.
- Consider the following example:
  SJMP LABEL_X



- An advantage of relative addressing is that the program code is easy to relocate in memory in that the addressing is relative to the position in memory.

❖ Indexed Addressing

- With indexed addressing a separate register, either the program counter, PC, or the data pointer DTPR, is used as a base address and the accumulator is used as an offset address.
- The effective address is formed by adding the value from the base address to the value from the offset address.
- Indexed addressing in the 8051 is used with the JMP or MOVC instructions. Look up tables are easy to implement with the help of index addressing.
- Consider the example instruction:
  MOVC A, @A+DPTR
- MOVC is a move instruction, which moves data from the external code memory space.
- The address operand in this example is formed by adding the content of the DPTR register to the accumulator value.
- Here the DPTR value is referred to as the base address and the accumulator value us referred to as the index address.

❖ Long Addressing

- The long addressing mode within the 8051 is used with the instructions LJMP and LCALL.
- The address specifies a full 16 bit destination address so that a jump or a call can be made to a location within a 64KByte code memory space (2_{16} = 64K).

- An example instruction is:

  LJMP 5000h ; full 16 bit address is specified in operand

❖ **Absolute addressing mode**

- Be used to specify the lowest 11 bits of the destination for the acall and ajmp instructions.
- Branching is limited within the current 2-KB page of the program memory: the upper 5 bits of the destination address are identical to the upper 5 bits of the current PC.
- Examples:

  acall subx ;calls the subroutine that is started at the label subx.

  ajmp next ;the instruction with the label next to be executed.

Bit Addressing modes:

- The 8051 MCU has a group of Boolean instructions that operate on bit operands.
- The internal data memory ranging from 20H to 2FH and many special function registers are also bit addressable.

❖ Bit Direct

In this addressing mode bit operations are done by specifying the bit address in the operand.

Examples:

SETB 7Fh

Clr78h

❖ Bit Inherent

In this addressing mode the name of the special function register whose bits are to be manipulated is used.

Examples:

SETB P0.0

MOV C, 00H

CLR PSW.7

Types of Instructions

The assembly level instructions include: data transfer instructions, arithmetic instructions, logical instructions, program control instructions, and some special instructions such as the rotate instructions.

❖ Data Transfer

Many computer operations are concerned with moving data from one location to another. The 8051 uses five different types of instruction to move data:

- MOV
- MOVX
- MOVC
- PUSH
- POP
- XCH

1. MOV:

- In the 8051 the MOV instruction is concerned with moving data internally, i.e. between Internal RAM, SFR registers, general registers etc.
- MOVX and MOVC are used in accessing external memory data.
- The MOV instruction has the following format:

  MOV destination <- source
- The instruction copies (copy is a more accurate word than move) data from a defined source location to a destination location.
- Example MOV instructions are:

```
MOV R2, #80h ; Move immediate data value 80h to register R2
MOV R4, A ; Copy data from accumulator to register R4
MOV DPTR, #0F22Ch ; Move immediate value F22Ch to the DPTR register
MOV R2, 80h ; Copy data from 80h (Port 0 SFR) to R2
MOV 52h, #52h ; Copy immediate data value 52h to RAM location 52h
MOV 52h, 53h ; Copy data from RAM location 53h to RAM 52h
MOV A,@R0 ; Copy contents of location addressed in R0 to A (indirect addressing)
```

2. MOVX:

- The 8051 the external memory can be addressed using *indirect* addressing only.
- The DPTR register is used to hold the address of the external data (since DPTR is a 16-bit register it can address 64KByte locations: $2^{16} = 64K$).

- The 8 bit registers R0 or R1 can also be used for indirect addressing of external memory but the address range is limited to the lower 256 bytes of memory (2^8 = 256 bytes).
- The MOVX instruction is used to access the external memory (X indicates eXternal memory access).
- All external moves must work through the A register (accumulator).

Examples of MOVX instructions are:

```
MOVX @DPTR, A ; Copy data from A to the address specified in DPTR
MOVX A, @DPTR ; Copy data from address specified in DPTR to A
```

3. MOVC:

- MOV instructions operate on RAM, which is (normally) a volatile memory. Program tables often need to be stored in ROM since ROM is nonvolatile memory. The MOVC instruction is used to read data from the internal code memory (ROM).
- DPTR register is used as the indirect address register.
- The indirect addressing is enhanced to realize an indexed addressing mode where register A can be used to provide an offset in the address specification.
- All moves must be done through register A.
- The following sequence of instructions provides an example:

```
MOV DPTR, # 2000h ; Copy the data value 2000h to the DPTR register
MOV A, #80h ; Copy the data value 80h to register A
MOVC A, @A+DPTR ; Copy the contents of the address 2080h (2000h + 80h)
; to register A
```

- Note, for the MOVC the program counter, PC, can also be used to form the address.

4. PUSH and POP:

- PUSH and POP instructions are used with the stack only.
- The SFR register SP contains the current stack address.
- Direct addressing is used as shown in the following examples:

```
PUSH 4Ch ; Contents of RAM location 4Ch is saved to the stack. SP is incremented.
PUSH 00h ;  The content of R0 (which is at 00h in RAM) is saved to the stack and
            SP is incremented.
POP 80h ;   The data from current SP address is copied to 80h and SP is
            decremented.
```

5. XCH:

- Move instructions copy data from a source location to a destination location, leaving the source data unaffected. A special XCH (eXCHange) instruction will actually swap the data between source and destination, effectively changing the source data.
- Immediate addressing may not be used with XCH. XCH instructions must use register A.
- XCHD is a special case of the exchange instruction where just the lower nibbles are exchanged.

- Examples using the XCH instruction are:
  XCH A, R3 ; Exchange bytes between A and R3
  XCH A, @R0 ; Exchange bytes between A and RAM location whose address is in R0
  XCH A, A0h ; Exchange bytes between A and RAM location A0h (SFR port 2)

❖ Arithmetic

Some key flags within the PSW, i.e. C, AC, OV, P, are utilized in many of the arithmetic instructions. The arithmetic instructions can be grouped as follows:

- Addition
- Subtraction
- Increment/decrement
- Multiply/divide
- Decimal adjust

1. Addition

- Register A (the accumulator) is used to hold the result of any addition operation.
- Some simple addition examples are:
  ADD A, #25h ; Adds the number 25h to A, putting sum in A
  ADD A, R3 ; Adds the register R3 value to A, putting sum in A
- The flags in the PSW register are affected by the various addition operations, as follows:
  - The C (carry) flag is set to 1 if the addition resulted in a carry out of the accumulator's MSB bit, otherwise it is cleared.
  - The AC (auxiliary) flag is set to 1 if there is a carry out of bit position 3 of the accumulator, otherwise it is cleared.
  - For signed numbers the OV flag is set to 1 if there is an arithmetic overflow (described elsewhere in these notes)
- Simple addition is done within the 8051 based on 8 bit numbers, but it is often required to add 16 bit numbers, or 24 bit numbers etc. This leads to the use of multiple byte (multi-precision) arithmetic. The least significant bytes are first added, and if a carry results, this carry is carried over in the addition of the next significant byte etc. This addition process is done at 8-bit precision steps to achieve multiprecision arithmetic.
- The ADDC instruction is used to include the carry bit in the addition process.
- Example instructions using ADDC are:
  ADDC A, #55h ; Add contents of A, the number 55h, the carry bit; and put the sum in A
  ADDC A, R4 ; Add the contents of A, the register R4, the carry bit; and put the sum in A.

2. Subtraction

- Computer subtraction can be achieved using 2's complement arithmetic.
- Most computers also provide instructions to directly subtract signed or unsigned numbers.
- The accumulator, register A, will contain the result (difference) of the subtraction operation.
- The C (carry) flag is treated as a borrow flag, which is always subtracted from the minuend during a subtraction operation.
- Some examples of subtraction instructions are:

```
SUBB A, #55d ; Subtract the number 55 (decimal) and the C flag from A; and put the
               result in A.
SUBB A, R6 ;   Subtract R6 the C flag from A; and put the result in A.
SUBB A, 58h ;  Subtract the number in RAM location 58h and the C flag
               From A; and put the result in A.
```

3. Increment/Decrement

- The increment (INC) instruction has the effect of simply adding a binary 1 to a number while a decrement (DEC) instruction has the effect of subtracting a binary 1 from a number.
- The increment and decrement instructions can use the addressing modes: direct, indirect and register.
- The flags C, AC, and OV are **not** affected by the increment or decrement instructions. If a value of FFh is increment it overflows to 00h. If a value of 00h is decrement it underflows to FFh.
- The DPTR can overflow from FFFFh to 0000h. The DPTR register cannot be decremented using a DEC instruction (unfortunately!).
- Some example INC and DEC instructions are as follows:

```
INC R7 ; Increment register R7
INC A ; Increment A
INC @R1 ; Increment the number which is the content of the address in R1
DEC A ; Decrement register A
DEC 43h ; Decrement the number in RAM address 43h
INC DPTR ; Increment the DPTR register
```

4. Multiply / Divide

- The 8051 supports 8-bit multiplication and division. This is low precision (8 bit) arithmetic but is useful for many simple control applications. The arithmetic is relatively fast since multiplication and division are implemented as single instructions. If better precision, or indeed, if floating point arithmetic is required then special software routines need to be written. For the MUL or DIV instructions the A and B registers must be used and only unsigned numbers are supported.

- **Multiplication**
  The MUL instruction is used as follows (note absence of a comma between the A and B operands):

  MUL AB ; Multiply A by B. The resulting product resides in registers A and B, the low-order byte is in A and the high order byte is in B.

- **Division**

The DIV instruction is used as follows:

DIV AB ; A is divided by B. The remainder is put in register B and the integer part of the quotient is put in register A.

5. Decimal Adjust (Special)

- The 8051 performs all arithmetic in binary numbers (i.e. it does not support BCD arithmetic).
- If two BCD numbers are added then the result can be adjusted by using the DA, decimal adjust, instruction:

  DA A ; Decimal adjust A following the addition of two BCD numbers.

❖ Logical

Boolean Operations

Most control applications implement control logic using Boolean operators to act on the data. Most microcomputers provide a set of Boolean instructions that act on byte level data. However, the 8051 (somewhat uniquely) additionally provides Boolean instruction which can operate on bit level data.

The following Boolean operations can operate on byte level or bit level data:

- ANL Logical AND
- ORL Logical OR
- CPL Complement (logical NOT)
- XRL Logical XOR (exclusive OR)

✦ Logical operations at the BYTE level

- The destination address of the operation can be the accumulator (register A), a general register, or a direct address.
- Status flags are not affected by these logical operations (unless PSW is directly manipulated).
- Example instructions are:

  ANL A, #55h ; AND each bit in A with corresponding bit in number 55h, leaving the result in A.

  ANL 42h, R4 ; AND each bit in RAM location 42h with corresponding bit in R4,

leaving the result in RAM location 42h.

ORL A,@R1 ; OR each bit in A with corresponding bit in the number whose address is contained in R1 leaving the result in A.

XRL R4, 80h ; XOR each bit in R4 with corresponding bit in RAM location 80h (port 0), leaving result in A.

CPL R0 ; Complement each bit in R0

✦ Logical operations at the BIT level

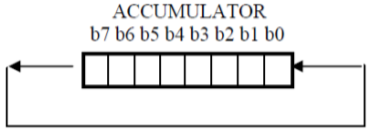
- The C (carry) flag is the destination of most bit level logical operations.
- The carry flag can easily be tested using a branch (jump) instruction to quickly establish program flow control decisions following a bit level logical operation.
- The following SFR registers only are addressable in bit level operations:
  - PSW
  - IE
  - IP
  - TCON
  - SCON
- Examples of bit level logical operations are as follows:

SETB 2Fh ;	Bit 7 of Internal RAM location 25h is set
CLR C ;	Clear the carry flag (flag =0)
CPL 20h ;	Complement bit 0 of Internal RAM location 24h
MOV C, 87h ;	Move to carry flag the bit 7of Port 0 (SFR at 80h)
ANL C,90h ;	AND C with the bit 0 of Port 1 (SFR at 90)
ORL C, 91h ;	OR C with the bit 1 of Port 1 (SFR at 90)

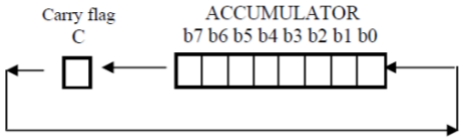
✦ Rotate Instructions

- The ability to rotate the A register (accumulator) data is useful to allow examination of individual bits. The options for such rotation are as follows:

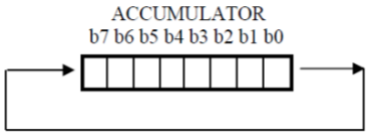
➢ RL A ; Rotate A one bit to the left. Bit 7 rotates to the bit 0 position



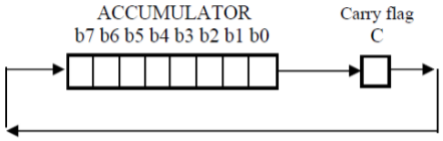
➢ RLC A ; The Carry flag is used as a ninth bit in the rotation loop



- RR A ; Rotates A to the right (clockwise)



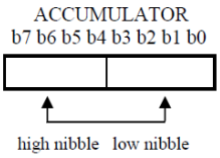
- RRC A ; Rotates to the right and includes the carry bit as the 9th bit.



✦ Swap = special

- The Swap instruction swaps the accumulator's high order nibble with the low-order nibble using the instruction:

SWAP A



✦ Program Control Instructions

- The 8051 supports three kind of jump instructions:
  - LJMP
  - SJMP
  - AJMP

LJMP

- LJMP (long jump) causes the program to branch to a destination address defined by the 16-bit operand in the jump instruction.
- Because a 16-bit address is used the instruction can cause a jump to any location within the 64KByte program space (2_{16} = 64K).
- Some example instructions are:

  LJMP LABEL_X ; Jump to the specified label
  LJMP 0F200h ; Jump to address 0F200h
  LJMP @A+DPTR ; Jump to address which is the sum of DPTR and Reg. A

SJMP

- SJMP (short jump) uses a single byte address.
- This address is a signed 8-bit number and allows the program to branch to a distance – 128 bytes back from the current PC address or +127 bytes forward from the current PC address.
- The address mode used with this form of jumping (or branching) is referred to as *relative addressing*, introduced earlier, as the jump is calculated relative to the current PC address.

AJMP

- This is a special 8051 jump instruction, which allows a jump with a 2KByte address boundary (a 2K page)
- There is also a generic JMP instruction supported by many 8051 assemblers.
- The assembler will decide which type of jump instruction to use, LJMP, SJMP or AJMP, so as to choose the most efficient instruction.

❖ Subroutines and program flow control

- A subroutine is called using the LCALL or the ACALL instruction.

LCALL

- This instruction is used to call a subroutine at a specified address.
- The address is 16 bits long so the call can be made to any location within the 64KByte memory space.
- When a LCALL instruction is executed the current PC content is automatically pushed onto the stack of the PC.
- When the program returns from the subroutine the PC contents is returned from the stack so that the program can resume operation from the point where the LCALL was made.
- The return from subroutine is achieved using the RET instruction, which simply pops the PC back from the stack.

ACALL

- The ACALL instruction is logically similar to the LCALL but has a limited address range similar to the AJMP instruction.
- CALL is a generic call instruction supported by many 8051 assemblers.
- The assembler will decide which type of call instruction, LCALL or ACALL, to use so as to choose the most efficient instruction.

Program control using conditional jumps

- Most 8051 jump instructions use an 8-bit destination address, based on relative addressing, i.e. addressing within the range –128 to +127 bytes.
- When using a conditional jump instruction the programmer can simply specify a program label or a full 16-bit address for the conditional jump instruction's destination.
- The assembler will position the code and work out the correct 8-bit relative address for the instruction.
- Some example conditional jump instructions are:

```
JZ LABEL_1 ;          Jump to LABEL_1 if accumulator is equal to zero
JNZ LABEL_X ;         Jump to LABEL_X if accumulator is not equal to zero
JNC LABEL_Y ;         Jump to LABEL_Y if the carry flag is not set
DJNZ R2, LABEL;       Decrement R2 and jump to LABEL if the resulting value of
                      R2 is not zero.
CJNE R1, #55h , LABEL_2 ; Compare the magnitude of R1 and the number 55h
                      and jump to LABEL_2 if the magnitudes are not
                      equal.
```

- Note, jump instructions such as DJNZ and CJNE are very powerful as they carry out a particular operation (e.g.: decrement, compare) and then make a decision based on the result of this operation.

8051 Data Type & Directives

The 8051 micro controller has only one data type. It is 8 bits, and the size of each register is also 8 bits. It is the job of the programmer to break down data larger than 8 bits (00 to FFH, or 0 to 255 in decimal) to be processed by the CPU. The data types used by the 8051 can be positive or negative.

DB (define byte)

- It is used to **define the 8-bit data**.
- When DB is used to define data, the numbers can be in decimal, binary, hex, or ASCII formats. For **decimal, the "D"** after the decimal number is **optional**, but using "**B" (binary) and "H" (hexadecimal)** for the others is required.
- To indicate **ASCII, simply place the characters in quotation marks** ('like this'). The assembler will assign the ASCII code for the numbers or characters automatically.
- The DB directive is the only directive that can be used to define ASCII strings larger than two characters; therefore, it should be used for all ASCII data definitions. Following are some DB examples:

```
ORG 2000H
```

```
DATA1:  DB   28   ; DECIMAL NUMBER 28
DATA2:  DB   1234h   ; HEXADECIMAL
DATA3:  DB   "ABCD"   ; ASCII CHARACTER
```

ORG (origin)

- The ORG directive is used to indicate the **beginning of the address**.
- The number that comes after ORG can be either in **hex or in decimal**.
- If the number is not followed by H, it is decimal and the assembler will convert it to hex.

EQU (equate)

- this is used to **define a constant without occupying a memory location**.
- The EQU directive does not set aside storage for a data item but associates a constant **value with a data label** so that when the label appears in the program, put constant value will be substituted for the label.
- the following uses EQU for the counter constant and then the constant is used to load the R3 register.

```
CONST   EQU   25H

MOV R3, #CONST   ; COPY 25 INTO R3
```

END

- This indicates to the **assembler the end of the source (asm) file**.
- The END directive is the last line of an 8051 program, meaning that in the **source code anything after the END directive is ignored by the assembler.**
- Some assemblers use **". END" (notice the dot)** instead of **"END".**