Microcontroller (EEC421)

Lecture 8

Dr. Islam Mohamed

Electrical Engineering Department
Shoubra Faculty of Engineering, Benha University
Islam.ahmed@fen.bu.edu.eg

Microcontroller

Lecture 8: ARITHMETIC OPERATIONS

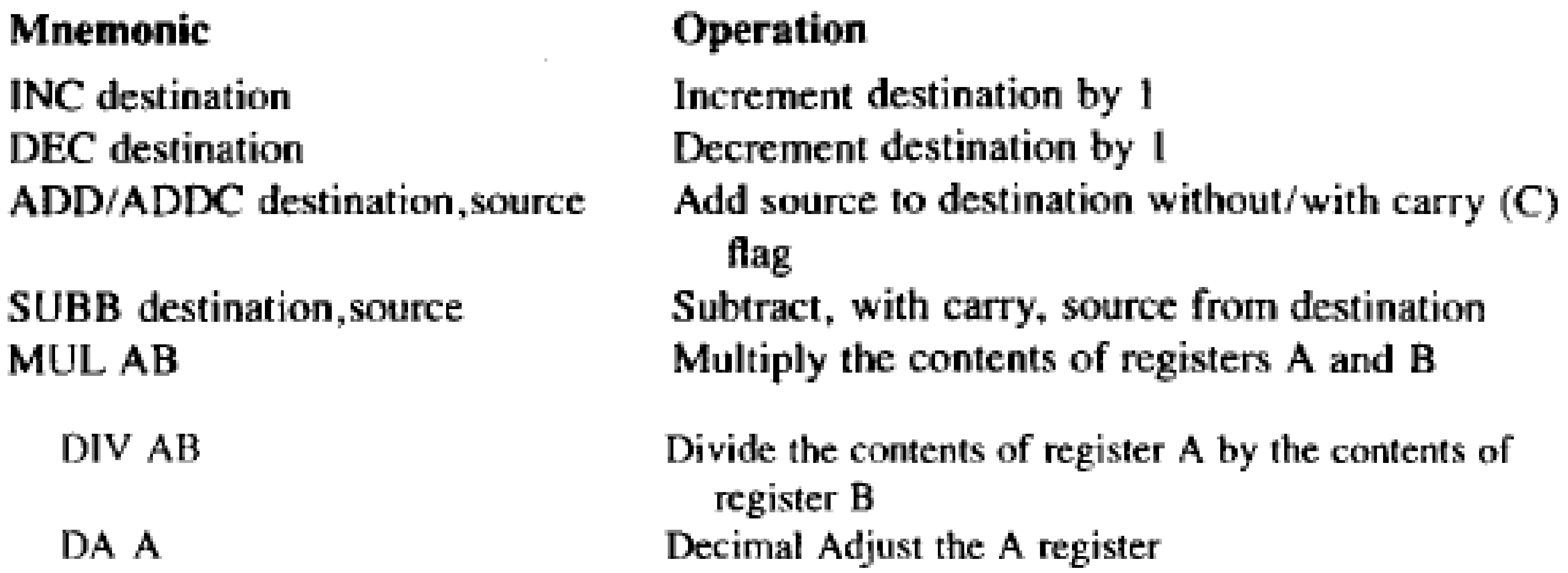
- ✓ Introduction
- ✓ Flags
- ✓ Instructions Affecting Flags
- ✓ Incrementing and Decrementing
- ✓ Addition :Unsigned and Signed Addition
- ✓ Multiple-Byte Signed Arithmetic
- ✓ Subtraction :Unsigned and Signed Subtraction
- ✓ Multiplication and Division
- ✓ Decimal Arithmetic

INTRODUCTION

- ✓ Applications of microcontrollers often involve performing mathematical calculations on data in order to alter program flow and modify program actions. The domain of the microcontroller is that of controlling events as they change (real-time control).
- ✓ A sufficient number of mathematical opcodes must be provided, however, so that calculations associated with the control of simple processes can be done, in real time, as the controlled system operates. When faced with a control problem, the programmer must know whether the 8051 has sufficient capability to expeditiously handle the required data manipulation. If it does not, a higher performance model must be chosen.

INTRODUCTION

- The arithmetic opcodes are grouped into the following types:



Mnemonic	Operation
INC destination	Increment destination by 1
DEC destination	Decrement destination by 1
ADD/ADDC destination,source	Add source to destination without/with carry (C) flag
SUBB destination,source	Subtract, with carry, source from destination
MUL AB	Multiply the contents of registers A and B
DIV AB	Divide the contents of register A by the contents of register B
DA A	Decimal Adjust the A register

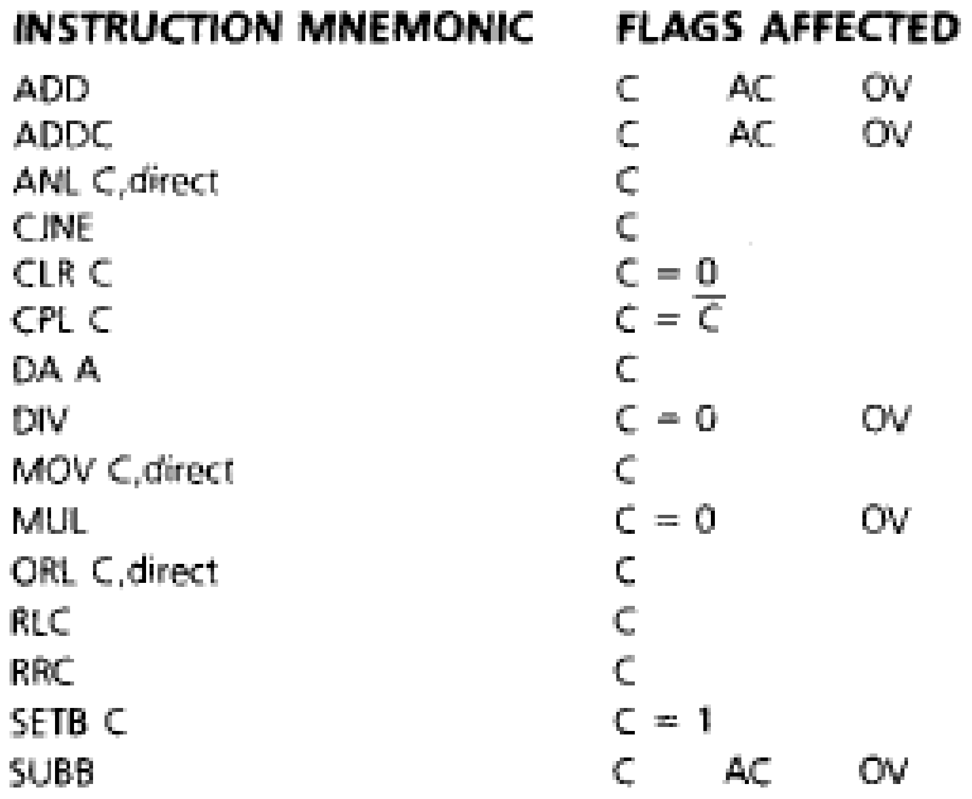
:

Flags

- A key part of performing arithmetic operations is the ability to store certain results of those operations that affect the way in which the program operates.
- For example, adding together two one-byte numbers results in a one-byte partial sum, because the 8051 is an eight-bit machine. But it is possible to get a 9-bit result when adding two 8-bit numbers.
- The ninth bit must be stored also, so the need for a one-bit register, or carry flag in this case, is identified. The program will then have to deal with the ninth bit, perhaps by adding it to a higher order byte in a multiple-byte addition scheme. Similar actions may have to be taken when a larger byte is subtracted from a smaller one. In this case, a borrow is necessary and must be dealt with by the program.
- Not all instructions change the flags, but many a programming error has been made by a forgetful programmer who overlooked an instruction that does change a flag.
- The 8051 has four arithmetic flags: the carry (C), auxiliary carry (AC), overflow (OV), and parity (P).

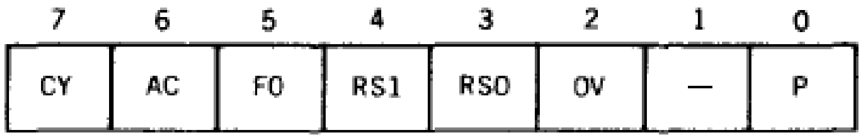
Instructions Affecting Flags

- ✓ The C. AC, and OV flags are arithmetic flags. They are set to 1 or cleared to 0 automatically, depending upon the outcomes of the following instructions.
- ✓ The following instruction set includes all instructions that modify the flags and is not confined to arithmetic instructions:



INSTRUCTION MNEMONIC	FLAGS AFFECTED		
ADD	C	AC	OV
ADDC	C	AC	OV
ANL C,direct	C		
CJNE	C		
CLR C	C = 0		
CPL C	C = $\overline{C}$		
DA A	C		
DIV	C = 0		OV
MOV C,direct	C		
MUL	C = 0		OV
ORL C,direct	C		
RLC	C		
RRC	C		
SETB C	C = 1		
SUBB	C	AC	OV

remember that the flags are all stored in the PSW.



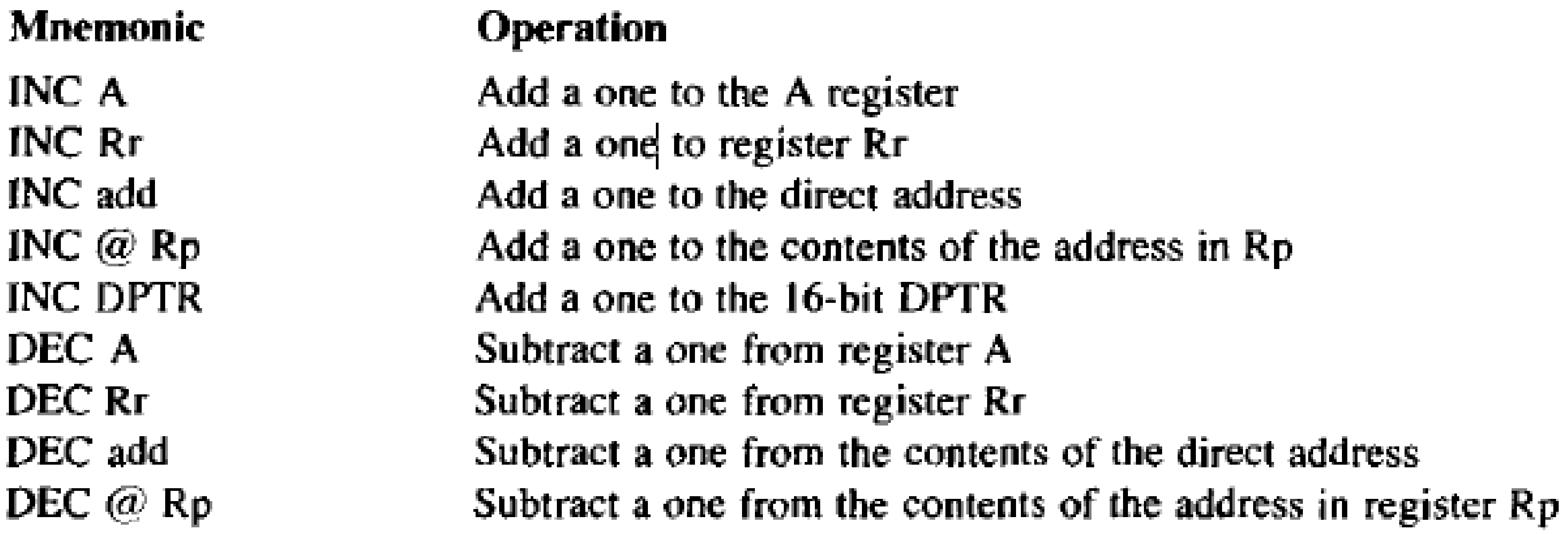
7	6	5	4	3	2	1	0
CY	AC	F0	RS1	RS0	OV	—	P

Instructions Affecting Flags

- ✓ A flag may be used for more than one type of result. For example, the C flag indicates a carry out of the lower byte position during addition and indicates a borrow during subtraction.
- ✓ The parity flag is affected by every instruction executed. The P flag will be set to a 1 if the number of 1 's in the A register is odd and will be set to 0 if the number of 1 's is even.

Incrementing and Decrementing

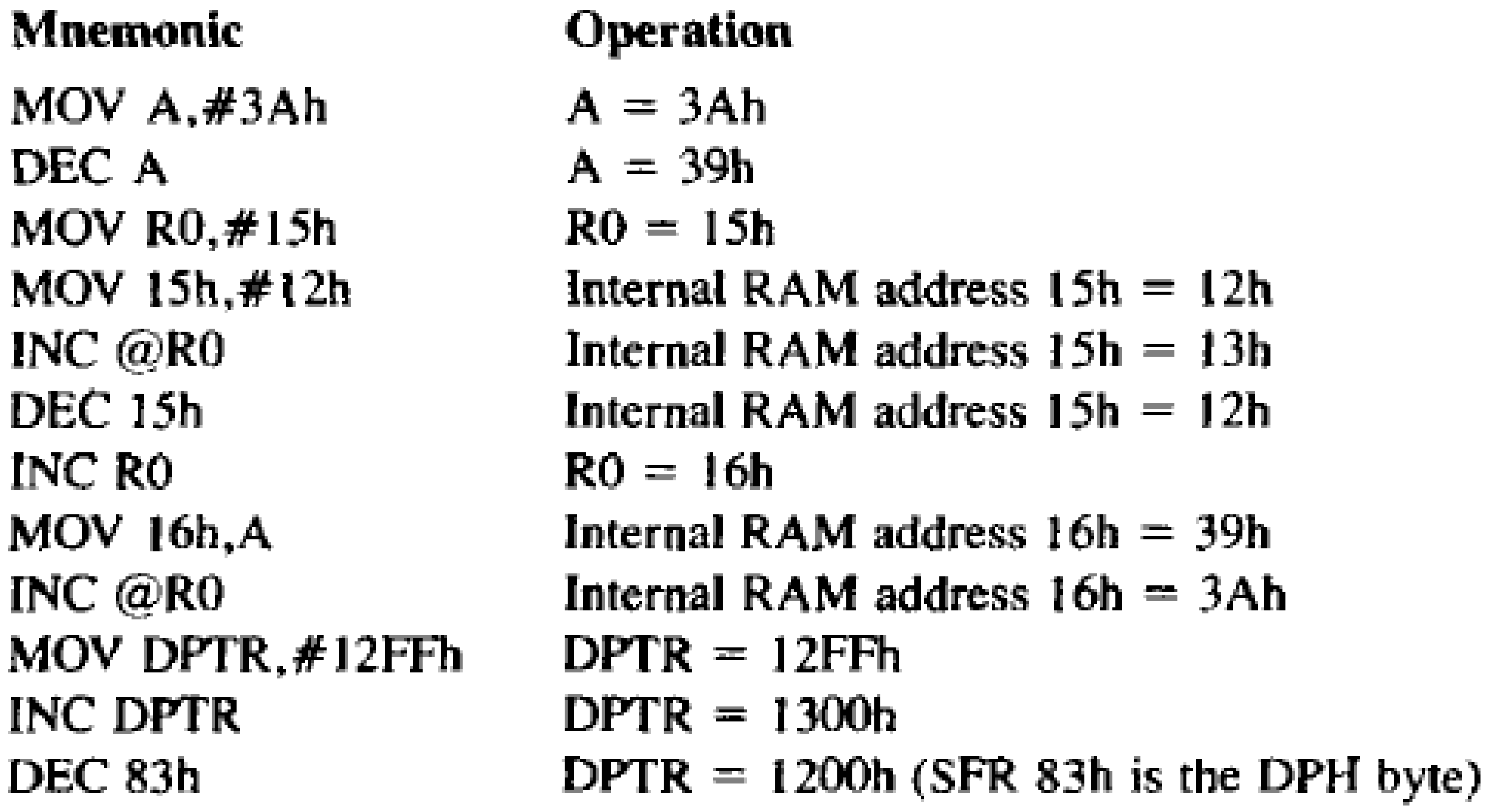
- ❖ The simplest arithmetic operations involve adding or subtracting a binary 1 and a number.
- ❖ These simple operations become very powerful when coupled with the ability to repeat the
- ❖ Operation that is, to "INCrement" or "DECrement" -until a desired result is reached.
- ❖ Register, Direct, and Indirect addresses may be INCremented or DECremented.
- ❖ No math flags (C, AC, OV) are affected.



Mnemonic	Operation
INC A	Add a one to the A register
INC Rr	Add a one to register Rr
INC add	Add a one to the direct address
INC @ Rp	Add a one to the contents of the address in Rp
INC DPTR	Add a one to the 16-bit DPTR
DEC A	Subtract a one from register A
DEC Rr	Subtract a one from register Rr
DEC add	Subtract a one from the contents of the direct address
DEC @ Rp	Subtract a one from the contents of the address in register Rp

Incrementing and Decrementing

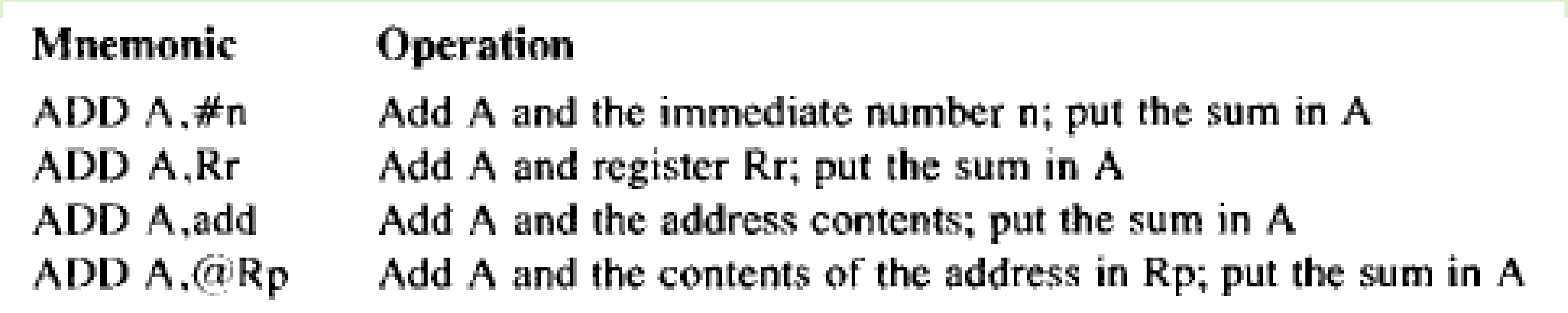
❖Example:



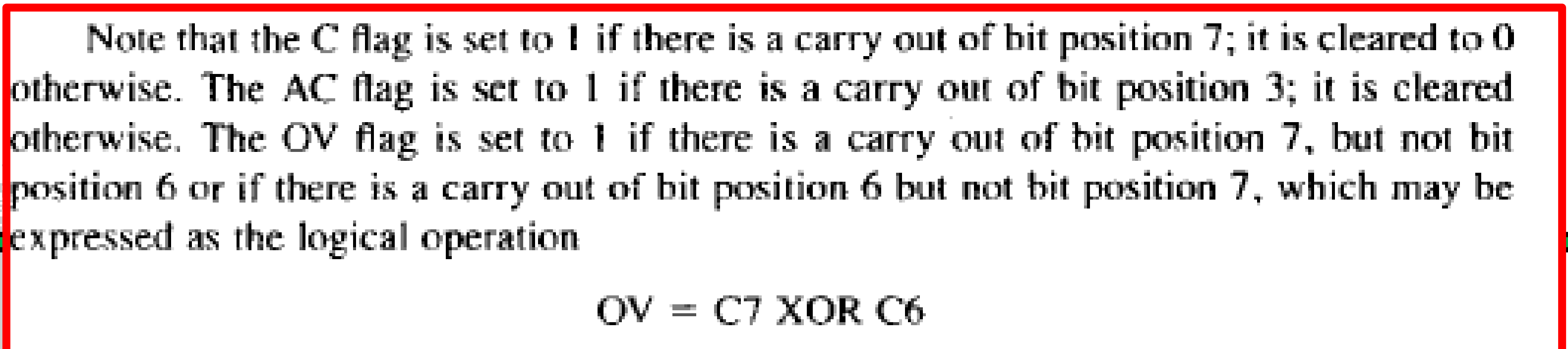
Mnemonic	Operation
MOV A,#3Ah	A = 3Ah
DEC A	A = 39h
MOV R0,#15h	R0 = 15h
MOV 15h,#12h	Internal RAM address 15h = 12h
INC @R0	Internal RAM address 15h = 13h
DEC 15h	Internal RAM address 15h = 12h
INC R0	R0 = 16h
MOV 16h,A	Internal RAM address 16h = 39h
INC @R0	Internal RAM address 16h = 3Ah
MOV DPTR,#12FFh	DPTR = 12FFh
INC DPTR	DPTR = 1300h
DEC 83h	DPTR = 1200h (SFR 83h is the DPH byte)

Addition

- All addition is done with the A register as the destination of the result.
- All addressing modes may be used for the source: an immediate number, a register, a direct address, and an indirect address.
- Some instructions include the carry flag as an additional source of a single bit that is included in the operation at the least significant bit position.



Mnemonic	Operation
ADD A,#n	Add A and the immediate number n; put the sum in A
ADD A,Rr	Add A and register Rr; put the sum in A
ADD A,add	Add A and the address contents; put the sum in A
ADD A,@Rp	Add A and the contents of the address in Rp; put the sum in A



Note that the C flag is set to 1 if there is a carry out of bit position 7; it is cleared to 0 otherwise. The AC flag is set to 1 if there is a carry out of bit position 3; it is cleared otherwise. The OV flag is set to 1 if there is a carry out of bit position 7, but not bit position 6 or if there is a carry out of bit position 6 but not bit position 7, which may be expressed as the logical operation

$$OV = C7 \text{ XOR } C6$$

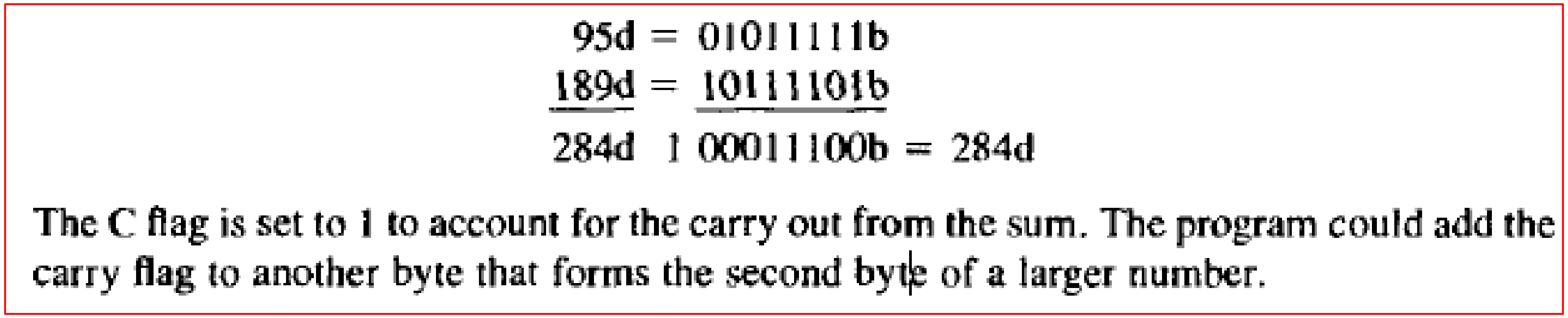
Unsigned and Signed Addition

- Signed numbers use bit 7 as a sign bit in the most significant byte (MSB) of the group of bytes chosen by the programmer to represent the largest number to be needed by the program.
- Bits 0 to 6 of the MSB, and any other bytes, express the magnitude of the number. Signed numbers use a 1 in bit position 7 of the MSB as a negative sign and a 0 as a positive sign. Further, all negative numbers are not in true form, but are in 2's complement form.
- In signed form, a single byte number may range in size from IOOOOOOOb, which is - 128d to 01111111 b, which is + 127d. The number OOOOOOOOb is OOOd and has a positive sign, so there are 128d negative numbers and 128d positive numbers.

Unsigned and Signed Addition

- Unsigned Addition

✓ Unsigned numbers make use of the carry flag to detect when the result of an ADD operation is a number larger than FFh. If the carry is set to one after an ADD, then the carry can be added to a higher order byte so that the sum is not lost. For instance,



```
  95d = 01011111b
 189d = 10111101b
 ----   ---------
 284d  1 00011100b = 284d
```

The C flag is set to 1 to account for the carry out from the sum. The program could add the carry flag to another byte that forms the second byte of a larger number.

Unsigned and Signed Addition

- Signed Addition

✓ If unlike signed numbers are added, then it is not possible for the result to be larger than -I28d or + 127d, and the sign of the result will always be correct.

✓ If positive numbers are added, there is the possibility that the sum will exceed + 127d, as demonstrated in the following example:

```
-001d = 11111111b
+027d = 00011011b
+026d   00011010b = +026d
```

```
+045d = 00101101b
+075d = 01001011b
+120d   01111000b = 120d
```

```
+100d = 01100100b
+050d = 00110010b
+150d   10010110b = -106d
```

✓ OV flag

Unsigned and Signed Addition

- Signed Addition

$$\begin{array}{rl} -070d = & 10111010b \\ \underline{-070d} = & \underline{10111010b} \\ -140d & 01110100b = +116d \end{array}$$

Or, the magnitude can be interpreted as −12d, which is the remainder after a carry out of −128d. In this example, there is a carry from bit position 7, and no carry from bit position 6, so the carry and the OV flags are set to 1. The magnitude of the sum is correct; the sign bit must be changed to a 1.

FLAGS		ACTION
C	OV	
0	0	None
0	1	Complement the sign
1	0	None
1	1	Complement the sign

A general rule is that if the OV flag is set, then complement the sign. The OV flag also signals that the sum exceeds the largest positive or negative numbers thought to be needed in the program.

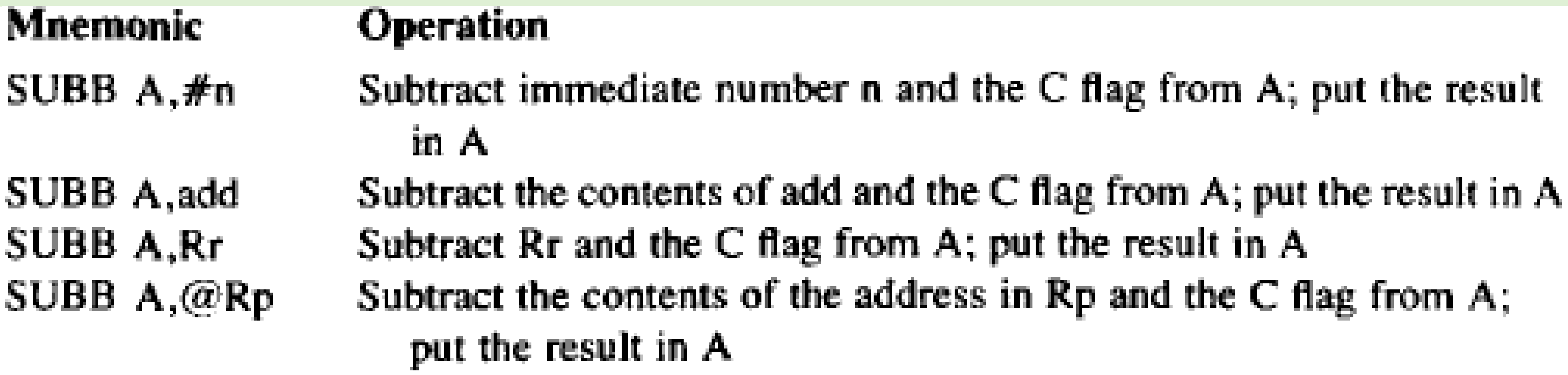
Add with Carry Mnemonics:

ADDC A,#n	Add the contents of A, the immediate number n, and the C flag; put the sum in A
ADDC A,add	Add the contents of A, the direct address contents, and the C flag; put the sum in A
ADDC A,Rr	Add the contents of A, register Rr, and the C flag; put the sum in A
ADDC A,@Rp	Add the contents of A, the contents of the indirect address in Rp, and the C flag; put the sum in A

Mnemonic	**Operation**
MOV A,#1Ch	A = 1Ch
MOV R5,#0A1h	R5 = A1h
ADD A,R5	A = BDh; C = 0, OV = 0
ADD A,R5	A = 5Eh; C = 1, OV = 1
ADDC A,#10h	A = 6Fh; C = 0, OV = 0
ADDC A,#10h	A = 7Fh; C = 0, OV = 0

Subtraction

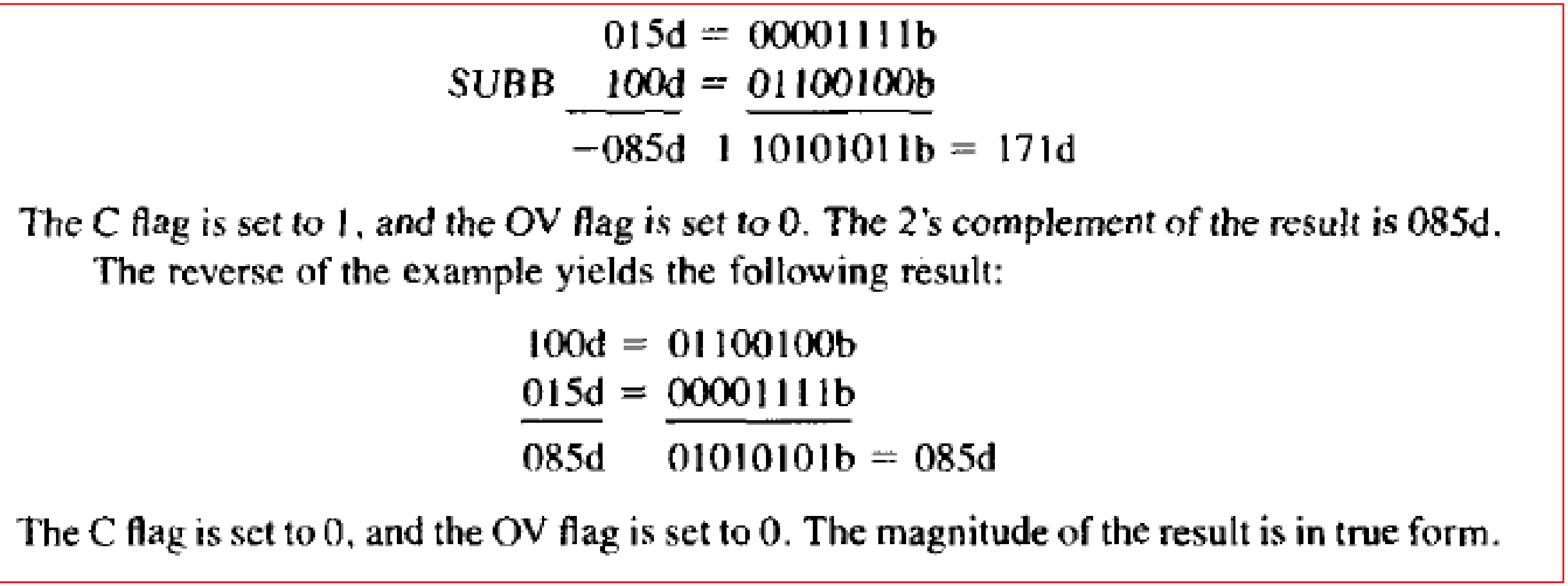
- Subtraction can be done by taking the 2's complement of the number to be subtracted, the subtrahend, and adding it to another number, the minuend. The 8051, however, has commands to perform direct subtraction of two signed or unsigned numbers.
- Register A is the destination address for subtraction. All four addressing modes may be used for source addresses. The commands treat the carry flag as a borrow and always subtract the carry flag as part of the operation.



Mnemonic	Operation
SUBB A,#n	Subtract immediate number n and the C flag from A; put the result in A
SUBB A,add	Subtract the contents of add and the C flag from A; put the result in A
SUBB A,Rr	Subtract Rr and the C flag from A; put the result in A
SUBB A,@Rp	Subtract the contents of the address in Rp and the C flag from A; put the result in A

Subtraction

Unsigned Subtraction



```
            015d = 00001111b
SUBB        100d = 01100100b
          ------   ---------
         −085d   1 10101011b = 171d
```

The C flag is set to 1, and the OV flag is set to 0. The 2's complement of the result is 085d.
The reverse of the example yields the following result:

```
100d = 01100100b
015d = 00001111b
----   ---------
085d   01010101b = 085d
```

The C flag is set to 0, and the OV flag is set to 0. The magnitude of the result is in true form.

Subtraction

Signed Subtraction

```
      +100d = 01100100b     (Carry flag = 0 before SUBB)
SUBB  +126d = 01111110b
      -----   ---------
      -026d 1 11100110b = -026d
```

There is a borrow into bit positions 7 and 6; the carry flag is set to 1, and the OV flag is cleared.

```
      -061d = 11000011b     (Carry flag = 0 before SUBB)
SUBB  -116d = 10001100b
      -----   ---------
      +055d   00110111b = +55d
```

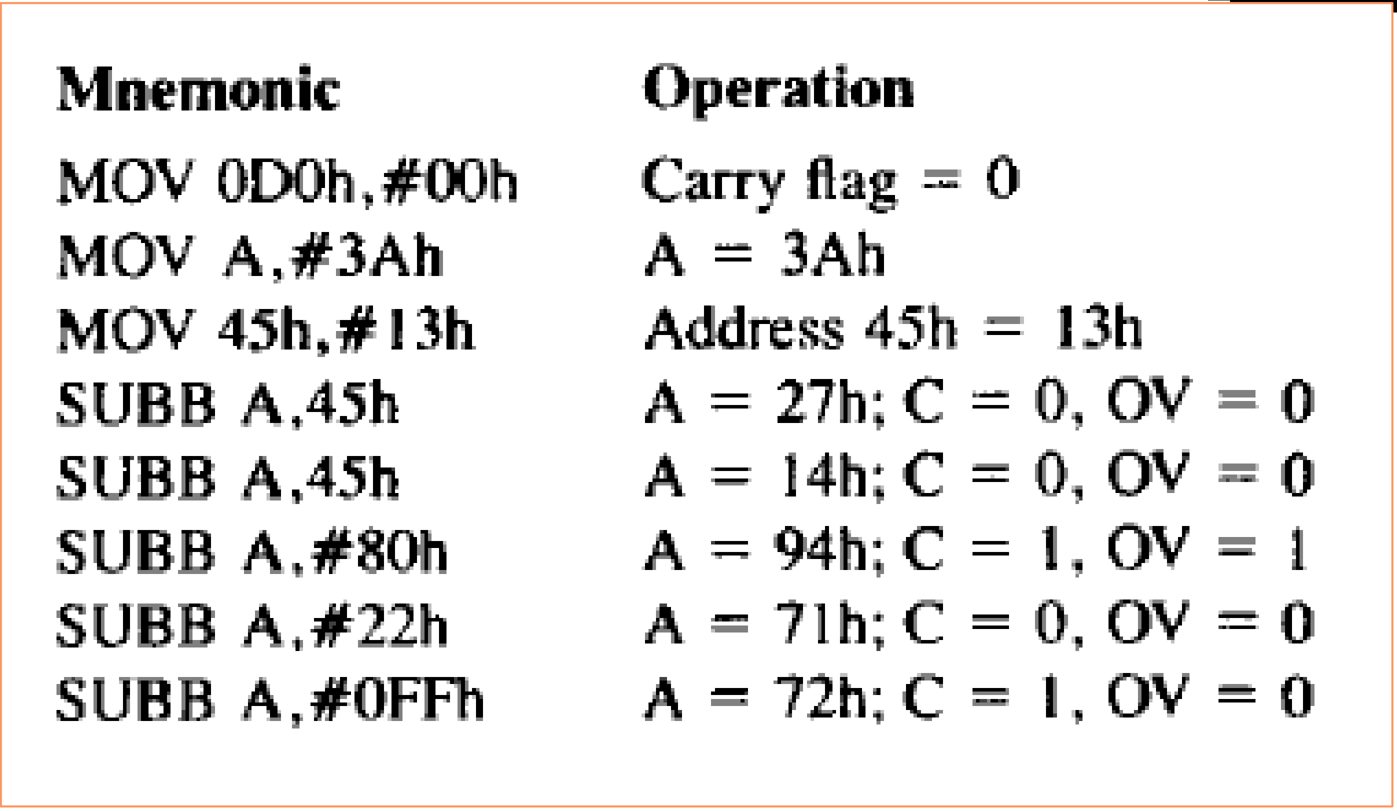
There are no borrows into bit positions 6 or 7, so the OV and carry flags are cleared to zero.

```
      -099d = 10011101b     (Carry flag = 0 before SUBB)
SUBB  +100d = 01100100b
      -----   ---------
      -199d   00111001b = +057d
```

Here, there is a borrow into bit position 6 but not into bit position 7; the OV flag is set to 1, and the carry flag is cleared to 0. Because the OV flag is set to 1, the result must be adjusted. In this case, the magnitude can be interpreted as the 2's complement of 71d, the remainder after a carry out of 128d from 199d. The magnitude is correct, and the sign needs to be corrected to a 1.

Subtraction

Example:



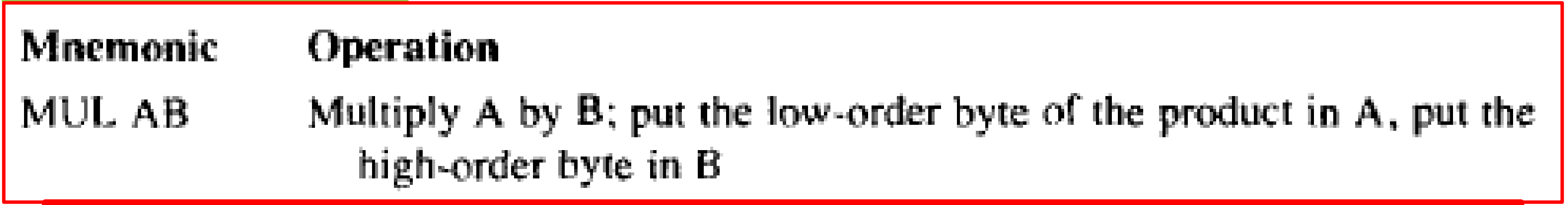
Mnemonic	Operation
MOV 0D0h,#00h	Carry flag = 0
MOV A,#3Ah	A = 3Ah
MOV 45h,#13h	Address 45h = 13h
SUBB A,45h	A = 27h; C = 0, OV = 0
SUBB A,45h	A = 14h; C = 0, OV = 0
SUBB A,#80h	A = 94h; C = 1, OV = 1
SUBB A,#22h	A = 71h; C = 0, OV = 0
SUBB A,#0FFh	A = 72h; C = 1, OV = 0

Multiplication and Division

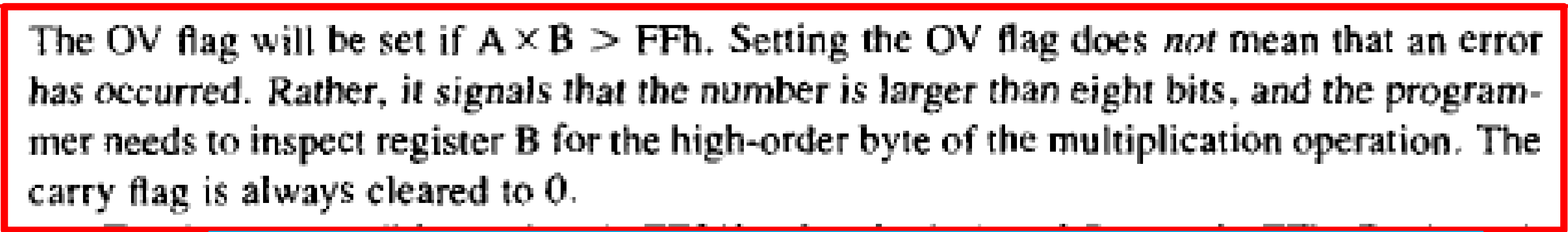
- The 8051 has the capability to perform 8-bit integer multiplication and division using the A and B registers. Register B is used solely for these operations and has no other use except as a location in the SFR space of RAM that could be used to hold data.
- The A register holds one byte of data before a multiply or divide operation, and one of the result bytes after a multiply or divide operation.
- Multiplication and division treat the numbers in registers A and B as unsigned. The programmer must devise ways to handle signed numbers.

Multiplication and Division

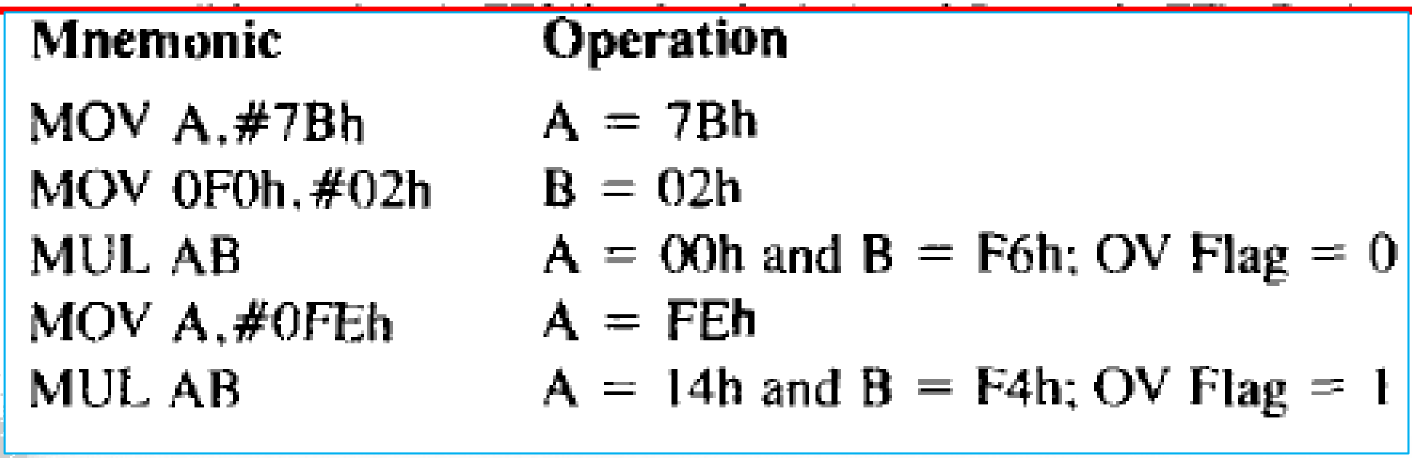
- Multiplication



Mnemonic	Operation
MUL AB	Multiply A by B; put the low-order byte of the product in A, put the high-order byte in B



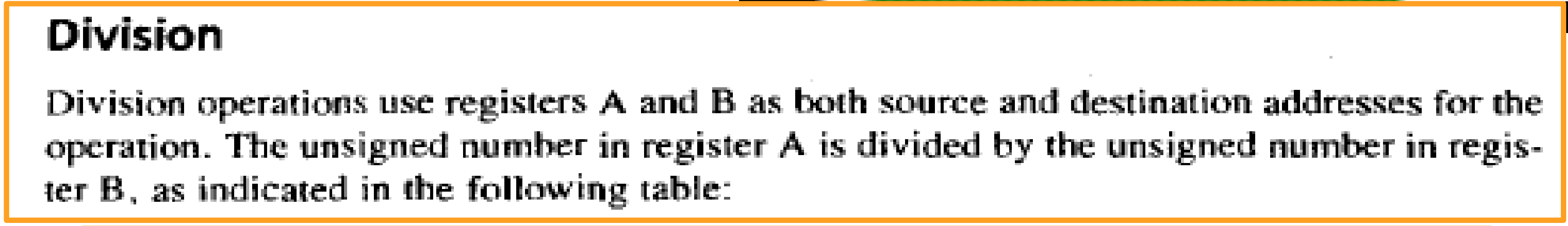
The OV flag will be set if A × B > FFh. Setting the OV flag does *not* mean that an error has occurred. Rather, it signals that the number is larger than eight bits, and the programmer needs to inspect register B for the high-order byte of the multiplication operation. The carry flag is always cleared to 0.



Mnemonic	Operation
MOV A,#7Bh	A = 7Bh
MOV 0F0h,#02h	B = 02h
MUL AB	A = 00h and B = F6h; OV Flag = 0
MOV A,#0FEh	A = FEh
MUL AB	A = 14h and B = F4h; OV Flag = 1

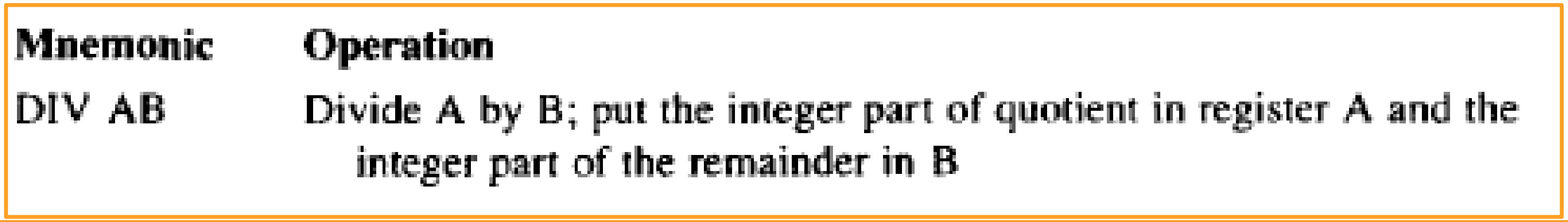
Multiplication and Division

- Division

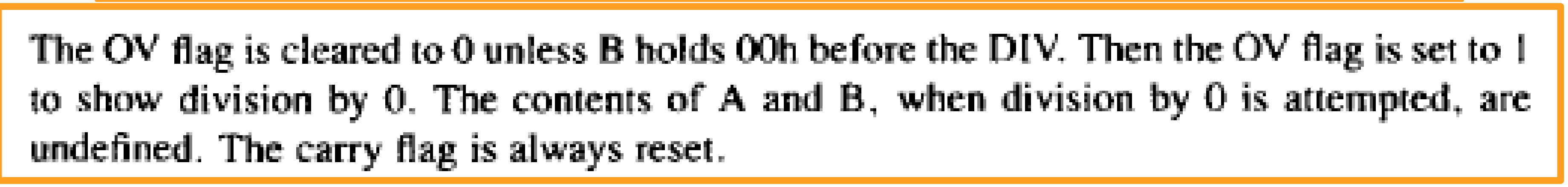


Division

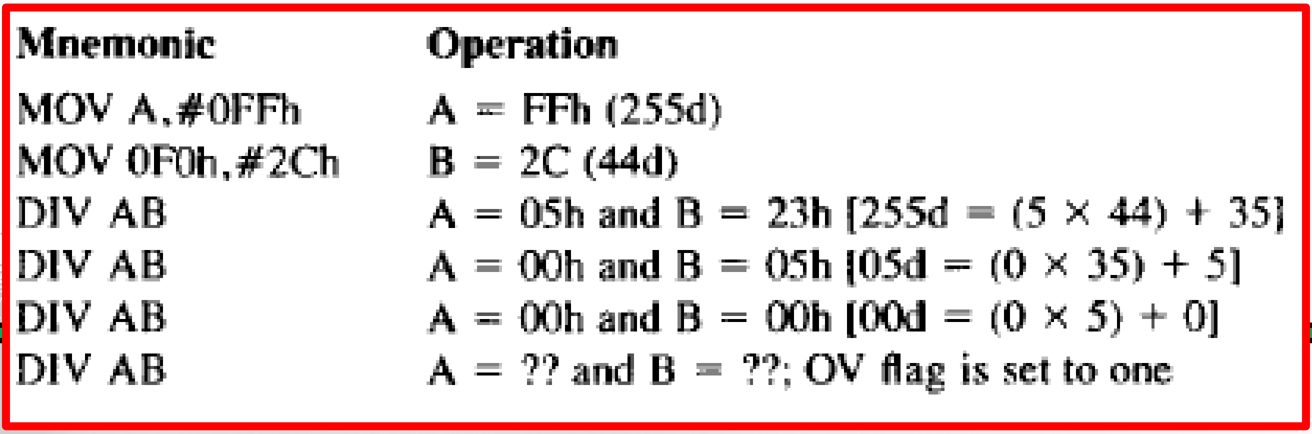
Division operations use registers A and B as both source and destination addresses for the operation. The unsigned number in register A is divided by the unsigned number in register B, as indicated in the following table:



Mnemonic	Operation
DIV AB	Divide A by B; put the integer part of quotient in register A and the integer part of the remainder in B



The OV flag is cleared to 0 unless B holds 00h before the DIV. Then the OV flag is set to 1 to show division by 0. The contents of A and B, when division by 0 is attempted, are undefined. The carry flag is always reset.



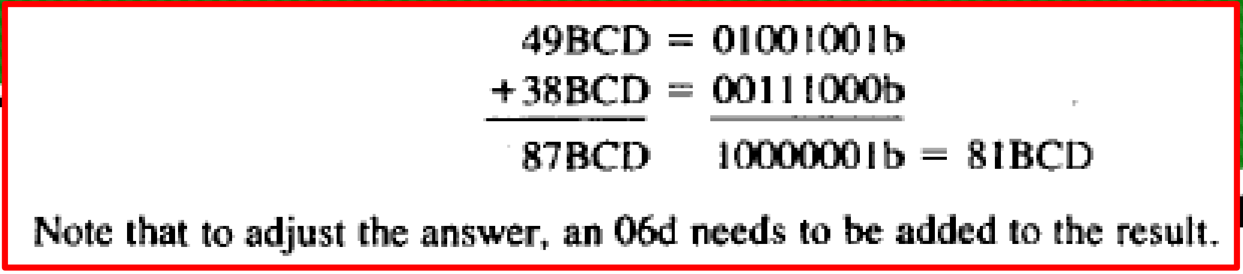
Mnemonic	Operation
MOV A,#0FFh	A = FFh (255d)
MOV 0F0h,#2Ch	B = 2C (44d)
DIV AB	A = 05h and B = 23h [255d = (5 × 44) + 35]
DIV AB	A = 00h and B = 05h [05d = (0 × 35) + 5]
DIV AB	A = 00h and B = 00h [00d = (0 × 5) + 0]
DIV AB	A = ?? and B = ??; OV flag is set to one

Decimal Arithmetic

- Most 8051 applications involve adding intelligence to machines where the hexadecimal numbering system works naturally. There are instances, however, when the application involves interacting with humans, who insist on using the decimal number system. In such cases, it may be more convenient for the programmer to use the decimal number system to represent all numbers in the program.

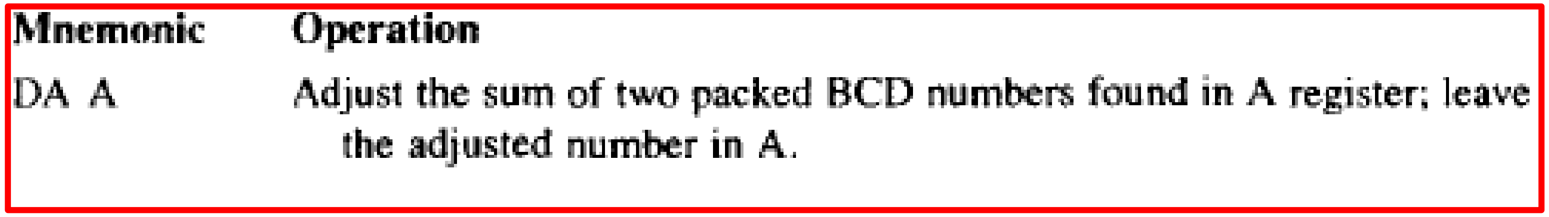
- Four bits are required to represent the decimal numbers from 0 to 9 (0000 to 1001) and the numbers are often called Binary coded decimal (BCD) numbers. Two of these BCD numbers can then be packed into a single byte of data. The 8051 does all arithmetic operations in pure binary. When BCD numbers are being used the result will often be a non-BCD number, as shown in the following example:

Decimal Arithmetic



```
  49BCD = 01001001b
 +38BCD = 00111000b
 ------   ---------
  87BCD   10000001b = 81BCD
```

Note that to adjust the answer, an 06d needs to be added to the result.



Mnemonic	Operation
DA A	Adjust the sum of two packed BCD numbers found in A register; leave the adjusted number in A.

- The C flag is set to I if the adjusted number exceeds 99BCD and set to 0 otherwise. It is important to remember that the DA A instruction assumes the added numbers were in BCD before the addition was done. Adding hexadecimal numbers and then using DA A will not convert the sum to BCD.
- The DA A opcode only works when used with ADD or ADDC opcodes and does not give correct adjustments for SUBB, MUL or DIV operations. The programmer might best consider the ADD or ADDC and DA A as a single instruction and use the pair automatically when doing BCD addition in the 8051.

Decimal Arithmetic

Mnemonic	Operation
MOV A,#42h	A = 42BCD
ADD A,#13h	A = 55h; C = 0
DA A	A = 55h; C = 0
ADD A,#17h	A = 6Ch; C = 0
DA A	A = 72BCD; C = 0
ADDC A,#34h	A = A6h; C = 0
DA A	A = 06BCD; C = 1
ADDC A,#11h	A = 18BCD; C = 0
DA A	A = 18BCD; C = 0