Arithmetic and Logical instructions

2. Arithmetic Instructions

Instructions of this group perform addition, subtraction, multiplication, division, increment, decrement, comparison, ASCII and decimal adjustment etc.

The following instructions come under this category:

Instruction	Description
ADD	Adds data to the accumulator i.e. AL or AX register or memory locations.
ADC	Adds specified operands and the carry status (i.e. carry of the previous stage).
SUB	Subtract immediate data from accumulator, memory or register.
SBB	Subtract immediate data with borrow from accumulator, memory or register.
MUL	Unsigned 8-bit or 16-bit multiplication.
IMUL	Signed 8-bit or 16-bit multiplication.
DIV	Unsigned 8-bit or 16-bit division.
IDIV	Signed 8-bit or 16-bit division.
INC	Increment Register or memory by 1.
DEC	Decrement register or memory by 1.
DAA	**Decimal Adjust after BCD Addition:** When two BCD numbers are added, the DAA is used after ADD or ADC instruction to get correct answer in BCD.
DAS	**Decimal Adjust after BCD Subtraction:** When two BCD numbers are added, the DAS is used after SUB or SBB instruction to get correct answer in BCD.
AAA	**ASCII Adjust for Addition:** When ASCII codes of two decimal digits are added, the AAA is used after addition to get correct answer in unpacked BCD.
AAD	**Adjust AX Register for Division:** It converts two unpacked BCD digits in AX to the equivalent binary number. This adjustment is done before dividing two unpacked BCD digits in AX by an unpacked BCD byte.
AAM	**Adjust result of BCD Multiplication:** This instruction is used after the multiplication of two unpacked BCD.
AAS	**ASCII Adjust for Subtraction:** This instruction is used to get the correct result in unpacked BCD after the subtraction of the ASCII code of a number from ASCII code another number.

CBW	Convert signed Byte to signed Word.
CWD	Convert signed Word to signed Doubleword.
NEG	Obtains 2's complement (i.e. negative) of the content of an 8-bit or 16-bit specified register or memory location(s).
CMP	Compare Immediate data, register or memory with accumulator, register or memory location(s).

3. Logical Instructions

Instruction of this group perform logical AND, OR, XOR, NOT and TEST operations.

The following instructions come under this category:

Instruction	Description
AND	Performs bit by bit logical AND operation of two operands and places the result in the specified destination.
OR	Performs bit by bit logical OR operation of two operands and places the result in the specified destination.
XOR	Performs bit by bit logical XOR operation of two operands and places the result in the specified destination.
NOT	Takes one's complement of the content of a specified register or memory location(s).
TEST	Perform logical AND operation of a specified operand with another specified operand.

Groups of the Arithmetic and Logic Instructions

Most Arithmetic and Logic Instructions affect the processor Flags register. There are 4 groups of Arithmetic and Logic Instructions based on operand:

1. ***First group of Arithmetic and Logic instructions have two operands***:

   Instructions: ADD, ADC, SUB, SBB, CMP, AND, TEST, OR, XOR

   Inst. operand1 , operand2

   After operation between two operands, the result of **ADD, ADC, SUB , SBB , AND, OR , XOR** instructions is always stored in ***first operand*** but **CMP** and **TEST** instructions affect flags only and do not store a result

in anywhere (these two instructions are used to make decisions during program execution).

These instructions affect these flags only: **CF, ZF, SF, OF, PF, AF.**

These types of operands are supported:

Instruction	***First operand*** , ***second operand*** **destination , Source**
ADD ADC SUB SBB CMP AND TEST OR XOR	REG, memory memory, REG REG, REG memory, immediate REG, immediate

REG 8-bit : AH, AL, BL, BH, CH, CL, DH, DL.

REG 16-bit :AX, BX, CX, DX, SI, DI, BP, SP, and only IP is never used as destination.

memory: [BX], [BX+SI+7], variable, etc...

immediate: 5, -24, 3Fh, 10001101b, etc...

ADD instruction: Add second operand to first.

Algorithm: operand1 = operand1 + operand2

Ex.

1. ADD AX, DX ; add the content of AX reg with DX reg then store the result in AX reg (AX = AX+DX)
2. ADD [SI], 1234 h ; add the content of memory location [SI] with immediate value 1234 h then store the result also in [SI] ([SI] = [SI]+1234 h)
3. Assume the content of F900 = 4120 h and BP = 321B h execute the instruction ADD BP,[F900]

   ADD BP,[F900] ; BP = BP + [F900]

   BP = 321B h + 4120 h = 733B h

ADC instruction: Add with carry.

Add the contents of the CF(0/1) to the first operand , and then adds the second operand to the first , just like ADD.

Algorithm:

operand1 = (operand1 + CF) + operand2

Ex.

STC ; set CF =1

MOV AL, 1C h AL = 1C h

OP1+CF= res1 , res1+op2=res2

ADC AL, 36 h AL = 53 h ; (1C+1=1D , 1D + 36 h= 53 h) ; 53=0101 0011, no.of 1's=4 4 means even; therefore PF=1.

CF=0 ; AF =1; PF = 1; ZF = 0 ; SF=0 ; OF = 0

The following instructions are wrong and not allowed:

Instructions	reasons
1. ADD DS, AX	; SREG
2. ADC BP, ES	; SREG
3. ADD [DI], CS	; SREG
4. ADC SS, 9000h	; SREG
5. SUB AX,DL	; 16 Bit with 8 bit
6. SBB [F400],66	; word ptr or byteptr
7. XOR 99, SP	; XOR SP,99
8. AND IP,6000	; IP is first operand
9. CMP [SI],[DI]	; memory with memory
10. TEST CX, DS	; SREG
11. OR BL, BX	; 8 bit with 16 bit

SUB instruction : Subtract second operand from first.

Algorithm:

operand1 = operand1 - operand2

Ex1. Write the instructions to subtract 03 from 0Ah. Then explain how the flag register changes.

MOV CL, 0A h ; CL=0A h

SUB CL , 3 ; CL = 07 (0A-03=07)

CF=0 ; AF = 0; SF=0 ; ZF =0 ; PF =0 ; OF =0

Or another way :

2'scomp operand2 (03) = 0000 0011=1111 1101=FD

operand1(0A) + 2'scomp operand2 (FD) = 0000 1010 +
1111 1101

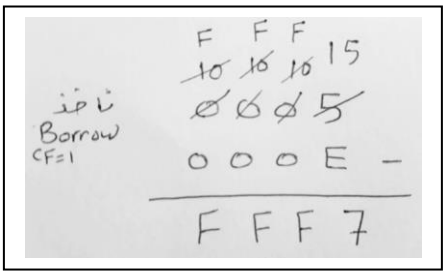
CF=1 0000 0 111 = 07

CF = 1 then 1's CF= 0

AF = 1 then 1's AF = 0

SF = 0 ; ZF= 0 ; PF = 0 ; OF =0

Ex2. Write the instructions to subtract 000Eh from 0005. Then explain how the flag register changes.



MOV AX, 05 h ; AX = 05

SUB AX , 0E ; AX = FFF7

CF=1 ; AF = 1; SF=1 ; ZF =0 ; PF =0 ; OF =0

SBB instruction : Subtract with Borrow. First subtracts the contents of the CF(0/1) from the first operand and then subtracts the second operand from the first , just like SUB.

Algorithm: operand1 = (operand1 – CF) - operand2

Ex.

STC

MOV CL,0B

OP1-CF= res1 , res1-op2=res2

SBB CL, 03 ; CL = 07 ; (0B-1= 0A , 0A-03=07)

CF=0 ; AF = 0; SF=0 ; ZF =0 ; PF =0 ; OF =0

Or another way : (2'scomp CF (01) = 0000 0001=1111 1111=FF)

operand1(0B) + 2'scomp CF (01) = 0000 1011+

1111 1111

0000 1010 = 0A res1

2'scomp operand 2 (03) = 0000 0011 = 1111 1101 = FD

res1(0A) + 2'scomp operand 2 (03) = 0000 1010 +

1111 1101

CF=1 0000 0111

CF = 1 then 1's CF= 0
AF = 1 then 1's AF = 0
SF = 0 ; ZF= 0 ; PF = 0 ; OF =0

CMP instruction: Compare internally subtract second operand from first **for flags only** (The result is not store in anywhere) ; means is the value of the first operand higher than , equal to , or lower than the value of the second operand?. The next section provides the various ways of transferring control (using conditional jump instructions) based on tested conditions according to flag values.

Algorithm:

operand1 - operand2 (the result is not store in anywhere)

The result of CMP instruction is one of the following:

- Operand 1 > operand 2
- Operand 1 = operand 2
- Operand 1 < operand 2

After CMP we use conditional jump instructions

Note : the result is not stored anywhere, flags are set (OF, SF, ZF, AF, PF, CF) according to result.

Ex.

```
MOV   AL,05
MOV   BL,05
CMP   AL, BL   ;  AL-BL  , but the result is not stored anywhere.
```

CF=0 , AF=0 , SF=0 , PF=0 , ZF = 1