Addr. Mode Implied Addressing mode

Operation AH → Lower byte of flag register

- This instruction copies the contents of AH register to the lower byte of flag register.
- It is included for 8085 compatibility.
- The OF, DF, IF and TF are not affected.

3. PUSHF – Push flags onto stack

(PUSH flag register on the stack)

Mnemonic PUSHF

Algorithm SP = SP – 2 **Flags** No flags are changed.

SS : [SP] (top of stack) = operand

Addr. Mode Register Addressing mode

Operation SP → SP – 2 SS → data from flag register.

- This instruction decrements the stack pointer by 2 and copies word in the flag register to the memory location pointed by stack pointer.
- The stack segment register is not affected.

4. POPF – Pop flags off stack

Mnemonic POPF

Algorithm SS = data to flag register **Flags** All flags are affected.

SP = SP + 2

Addr. Mode Register Addressing mode

Operation SS : [SP] → Copy data to flag register SP = SP + 2

- This instruction copies a word from the two memory locations at the top of the stack to flag register and increments the stack pointer by 2.
- The stack segment register and word on the stack are not affected.

12.7 Arithmetic Instructions

- Table 12.7.1 lists out arithmetic instructions available in 8086 microprocessor, under these subgroups.

Push BX

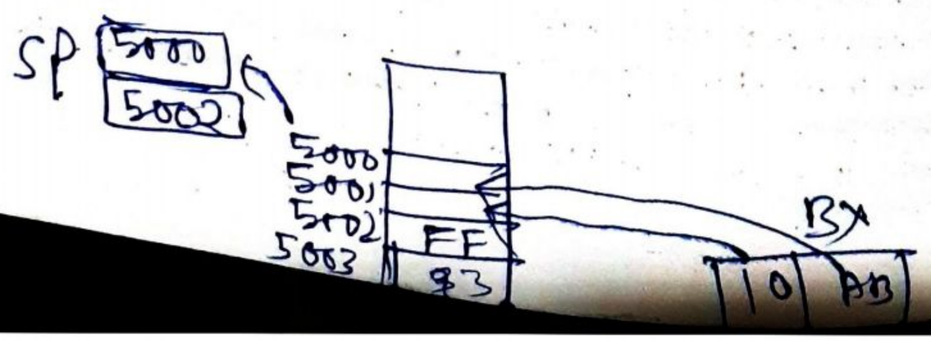
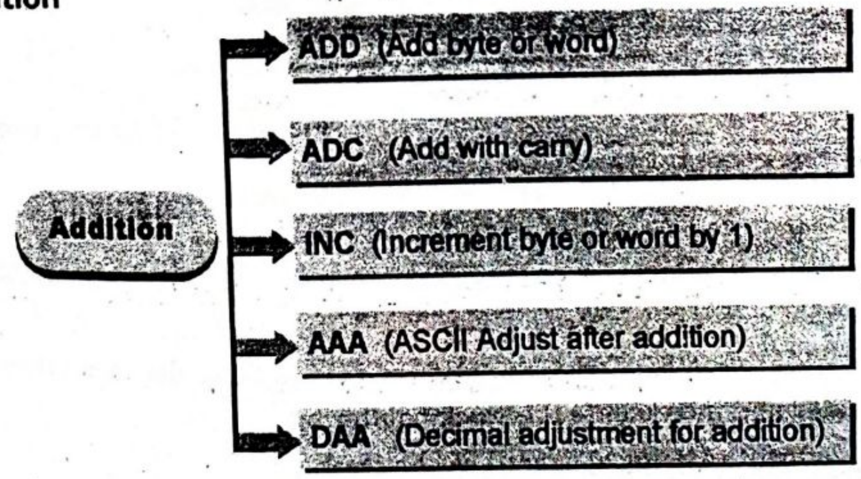


Table 12.7.1 : Arithmetic instructions

Addition		Multiplication	
ADD	Add byte or word	MUL	Multiply byte or word unsigned
ADC	Add byte or word with carry	IMUL	Multiply byte or word signed
INC	Increment byte or word by 1	AAM	Integer multiply byte or word
AAA	ASCII adjust for addition		ASCII adjust for multiply
DAA	Decimal adjustment for addition		
Subtraction		**Division**	
SUB	Subtract byte or word	DIV	Divide byte or word unsigned
SBB	Subtract byte or word with borrow	IDIV	Integer divide byte or word
DEC	Decrement byte or word by 1	AAD	ASCII adjust for division
NEG	Negate byte or word (2's complement)	CBW	Convert byte to word
CMP	Compare byte or word	CWD	Convert word to doubleword
AAS	ASCII adjust for subtraction		
DAS	Decimal adjust for subtraction		

We will study the subgroups one by one.

12.7.1 Addition


Addition
ADD (Add byte or word)
ADC (Add with carry)
INC (Increment byte or word by 1)
AAA (ASCII Adjust after addition)
DAA (Decimal adjustment for addition)

1. ADD – Add byte or word

Mnemonic : ADD destination, source

Algorithm : destination = destination + source

Operation : (Destination) $\leftarrow$ (destination) + (source)

This instruction adds a number from source to number from destination and puts the result to specified destination.

Sr. No.	Destination	Source
1.	Register	Register
2.	Register	Memory
3.	Memory	Register
4.	Register	Immediate
5.	Memory	Immediate
6.	Accumulator	Immediate

Note : (i) Both the operands i.e. source and destination, cannot be memory locations.
e.g. ADD [1234], [5678] ⇒ Incorrect
(ii) Source and destination both have to be of same type i.e. byte or word.

2. ADC – Add with carry

Mnemonic ADC destination, source. **Flags** All flags are affected.

Algorithm destination = destination + source + CY

Addr. Mode Register Immediate addressing mode.

Operation destination ← destination + source + CY

- This instructions adds destination operand contents, source operand contents and carry flag and answer is stored back to destination operand.
- The source and destination can be 8/16 bit register or memory location . The source and destination can also be a 8/16 bit register or memory location and immediate data .
- The segment registers cannot be used . The memory uses DS as segment register .
- The addition of two memory locations along with carry is not possible .
- It is easy to perform multiple- precision arithmetic by using ADC instruction.

Examples

1. ADC AL, BL i.e. AL ← AL + BL + CY	• This instruction adds the contents of AL register with BL register and contents of CY.
2. ADD BL, CL BL ← BL + CL	• This instruction adds the data in register CL and BL. The result is stored in BL register. It can be 8 bit/ 16 bit instruction
3. ADD AX , [2048] i.e. AX ← AX + contents of memory location	• This instruction adds the data at memory locations whose offset in DS are [2048] and [2049] with data in AX register. The result is stored in the AX register.

4. ADD [2048], AX	• This instruction adds the data at memory locations whose offset in DS are [2048] and [2049] with data in AL register and AH register.
5. ADD AL, 74 H i.e. AL ← AL + 74	• This instruction adds the immediate number 74 H with the contents of AL register . The result is stored in AL.

3. INC – Increment byte or word by 1

Mnemonic INC Destination **Flags** All the flags except carry flag are affected.

Algorithm destination = destination + 1

Addr. Mode Implied addressing mode

Operation destination ← destination + 1

- This instruction adds 1 to the destination operand.
- The operand may be a byte or word and is treated as an unsigned binary number.
- The destination operand may be a register or memory location.

Example	INC CX	Add 1 to contents of CX
	INC AL	Add 1 to contents of AL register.

4. AAA – ASCII Adjust after addition

Mnemonic AAA **Flags** AF and CF flags are changed, while OF, PF, SF, ZF are left unchanged.

Algorithm
```
If lower nibble of AL > 9 or AF = 1   then : AL = AL + 6
AH = AH + 1          AF = 1           CF = 1
else : AF = 0 , CF = 0
In both cases, clear the higher nibble of AL.
```

Addr. Mode Implied addressing mode

Operation

- Numerical data coming into a computer from a terminal through keyboard is usually in ASCII code. The numbers 0 to 9 are represented by ASCII codes 30 H to 39 H. The 8086 allows to add the ASCII codes for two decimal digits without masking off "3" in the upper nibble of each. After addition, AAA instruction is used to make sure that the result is the correct unpacked BCD.
- The AAA instruction works only on AL register.
- If the lower nibble of AL register after addition is greater than 9, add 6 to it and increment AH by 1. The auxiliary carry flag and carry flag are set.

Example Assume

AL = 0 0 1 1 0 1 0 1 ASCII 5

BL = 0 0 1 1 1 0 0 1 ASCII 9

ADD AL, BL

$$\begin{array}{lcccccccccl}
 & & 0&0&1&1 & 0&1&0&1 & \\
 & + & 0&0&1&1 & 1&0&0&1 & \\
\hline
 & & 0&1&1&0 & 1&1&1&0 & \rightarrow \text{Result of addition} \\
 & + & & & & & 0&1&1&0 & \text{Invalid BCD}\\
\hline
\text{AAA} & & 0&0&0&0 & \underbrace{0\;1\;0\;0}_{\Downarrow} & & & & \\
\text{Carry} \rightarrow & \boxed{1} & \multicolumn{4}{l}{\text{Clear higher nibble}} & & & & &
\end{array}$$

(04 H) → Result in AL valid BCD.

(01 H) → Result in AH. The carry is stored in AH register

5. DAA – Decimal adjustment for addition

Mnemonic DAA **Flags** It changes AF, CF, PF, ZF and SF.

Algorithm If lower nibble of AL > 9 or AF = 1 then, AL = AL + 06 H, AF = 1

If AL > 9F H or CF = 1 then, AL = AL + 60 H, CF = 1

Addr. Mode Implied addressing mode

Operation AL ← Sum in adjusted to packed BCD format

- This instruction is used to make sure that the result of adding two packed BCD numbers is adjusted to be a valid BCD number.
- It operates only on AL register.
- If number in the lower nibble of AL register after addition is greater than 9 or if the auxiliary carry flag is set add 6.
- If the lower nibble of AL is greater than 9 or if the carry flag is set then add 60 H.

Example If AL = 59 H valid BCD, BL = 34 H valid BCD

ADD AL, BL

$$\begin{array}{rcc} & 0\ 1\ 0\ 1 & 1\ 0\ 0\ 1 \\ + & 0\ 0\ 1\ 1 & 0\ 1\ 0\ 0 \\ \hline & \underbrace{1\ 0\ 0\ 0}_{8} & \underbrace{1\ 1\ 0\ 1}_{D} \end{array}$$

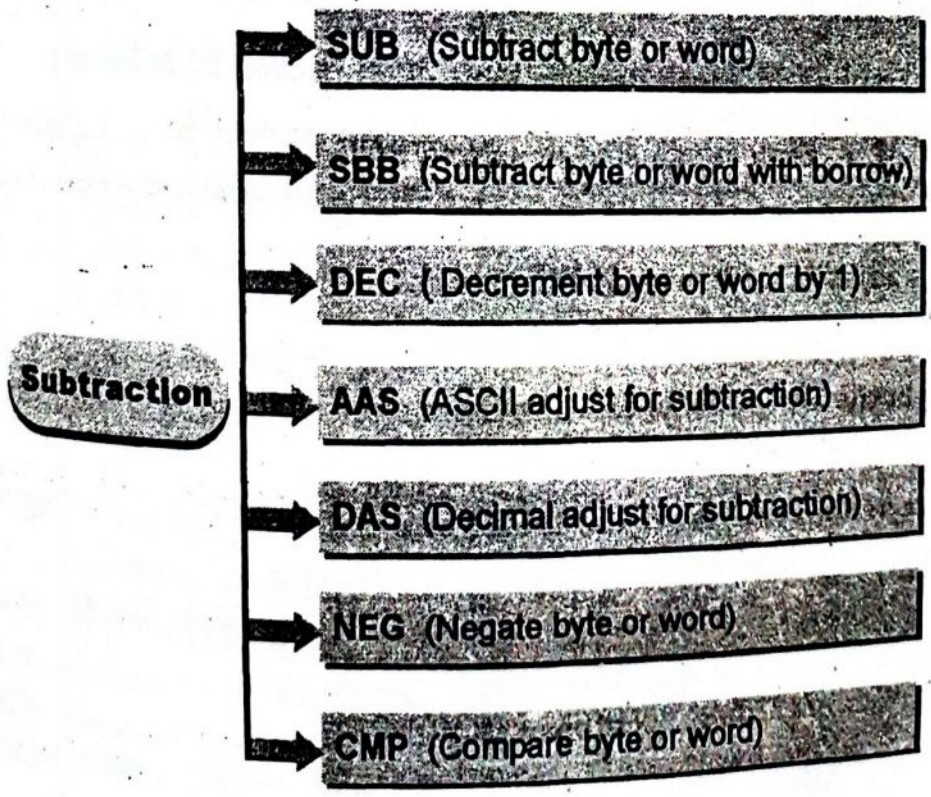
AL = 8DH invalid BCD after addition of AL and BL

DAA →

$$\begin{array}{rcc} & 1\ 0\ 0\ 0 & 1\ 1\ 0\ 1 \\ + & 0\ 0\ 0\ 0 & 0\ 1\ 1\ 0 \\ \hline & \underbrace{1\ 0\ 0\ 1}_{9} & \underbrace{0\ 0\ 1\ 1}_{3} \end{array}$$

∴ AL = 93 H BCD

12.7.2 Subtraction


Subtraction
SUB (Subtract byte or word)
SBB (Subtract byte or word with borrow)
DEC (Decrement byte or word by 1)
AAS (ASCII adjust for subtraction)
DAS (Decimal adjust for subtraction)
NEG (Negate byte or word)
CMP (Compare byte or word)

1. SUB – Subtract byte or word

Mnemonic SUB destination, source

Flags The flags affected are AF, CF, OF, PF, SF and ZF.

Algorithm destination = destination – source

Addr. Mode –

Operation destination ← destination – source

- This instruction subtracts a number from source with number from destination and puts result in destination location.
- Both the operands i.e. source and destination, cannot be memory locations.

Examples

1. SUB BL, CL BL ← BL – CL	This instruction subtracts the contents of CL register from BL and result is stored in BL.
2. SUB AX , [2048]	AX ← AX – contents of memory location • This instruction subtracts the data at memory locations whose offset in DS are [2048] and [2049] with data in AX register. • The result is stored in the AX register.
3. SUB [2048], AX	[2048] ← [2048] – AL, [2049] ← [2049] – AH • This instruction subtracts the data at memory locations whose offset in DS are [2048] and [2049] with data in AL register and AH register. • The result is stored at memory locations whose offset in DS are [2048] and [2049] i.e. (25188 H and 25189 H).
4. SUB COST, 14 H. COST ← COST–14 H [Cost : is a memory location]	• This instruction will subtract the immediate Data 14 H with contents of memory location COST whose offset is given in the instruction [i.e. memory location 28354 H] • Result of subtraction is stored at COST.
5. SUB AL, 24 H AL ← AL – 24	• This instruction subtracts the immediate number 24 H with the contents of AL register. The result is stored in AL.

The type of both the operands should match i.e. byte or word.

Destination	Source
Register	Register
Register	Memory
Memory	Register
Accumulator	Immediate
Register	Immediate
Memory	Immediate

2. SBB – Subtract byte or word with borrow

Mnemonic	SBB destination, source	**Flags**	SF flag is not affected.
Algorithm	Destination = destination – source – CY		
Addr. Mode	Register addressing mode.		
Operation	Destination – source – CY		

- This instruction subtracts source and carry flag (i.e. Borrow) from destination.
- The source and destination can be 8/16 bit register or memory location . The source and destination can also be a 8/16 bit register or memory location and immediate data .
- The segment registers cannot be used . The memory uses DS as segment register .
- The result is stored in destination.
- Both the source and destination may be words or bytes.
- They can be signed or unsigned binary numbers.
- It can be used to write routines that subtract the numbers longer than 16 bits i.e. double words ,quad words etc.

Example SBB AL, BL AL ← AL – BL – CY

- This instruction subtracts contents of BL and CY from AL.
- The result is stored in AL.

3. DEC – Decrement byte or word by 1

Mnemonic	DEC destination	**Flags**	CF is not affected
Algorithm	destination = destination – 1		
Addr. Mode	Register Addressing mode		
Operation	destination ← destination – 1		

- This instruction subtracts 1 from the destination word or byte.
- The destination can be a register or a memory location.
- This instruction cannot be used to decrement the contents of segment registers.

Example DEC AL AL = AL – 1

- This instruction subtracts 1 from the contents of CL and result is stored in CL.

4. AAS – ASCII adjust for subtraction

Mnemonic AAS

Flags It updates AF, CF but OF, PF, SF and ZF are left undefined.

Algorithm If lower nibble of AL > 9 or AF = 1 then :
AL = AL – 06 H AH = AH – 1 AF = 1 CF = 1
else AF = 0 CF = 0
In both cases clear the high nibble of AL

Addr. Mode Implied addressing mode

Operation

- The number 0 to 9 are represented as 30 – 39 in ASCII code
- The 8086 allows us to subtract the ASCII codes for two decimal digits without masking "3" in upper nibble of each.
- The AAS instruction is used to make sure that the result is in unpacked BCD form.
- It works only on the AL register.

Example Let AL = 0011 1001 = 39 H = ASCII 9

Let BL = 0011 0101 = 35 H = ASCII 5

SUB AL, BH → Result AAS	=	0011 1001	
	–	0011 0101	
	0	0000 0100	= BCD = 04
CF = 0 no borrow required			

5. DAS – Decimal adjust for subtraction

Mnemonic DAS.

Algorithm If lower nibble of AL > 9 or AF = 1 then **Flags** The OF is undefined.

AL = AL – 06 H AF = 1

If AL > 9F H or CF = 1 then AL = AL – 60 H

CF = 1

Addr. Mode Implied Addressing mode

Operation AL ← difference in AL adjusted to packed BCD format

- This instruction is used after subtracting two packed BCD numbers to make sure that the result is correct packed BCD. The result of subtraction must be in AL for DAS to work correctly.
- This instruction works only on AL register.
- If the lower nibble in AL after a subtraction is greater than 9 or if the auxiliary carry flag is set then the DAS instruction will subtract 6 from the lower nibble of AL.
- If the result in the upper nibble is greater than 9 or the carry flag is set the DAS instruction will subtract 60 H from the AL register

Example

AL = 1000 0110 = 86 BCD

BH = 0101 0111 = 57 BCD

SUB AL, BH

DAS

AL = 0010 1111 = 2F H, CF = 0.

Lower nibble of result is F H, i.e. greater than 9 so DAS subtracts 0000 0110 to give AL = 0010 1001 = 29 H BCD

6. NEG – Negate byte or word

Mnemonic NEG destination **Flags** All flags are affected.

Algorithm Invert all bits of operand and add 1 to inverted operand

Addr. Mode Register Addressing mode

Operation

- This instruction replaces the number is a destination with 2's complement of that number
- The destination can be a register or memory location.
- This instruction forms the 2's complement by subtracting the original word or byte in the indicated destination from zero.
- It is useful for changing the sign of a signed word or byte.
- Attempt to negate a byte containing – 128 or a word containing – 32, 768 causes no change to the operand and sets Overflow flag.

Example AX ← 2's complement of number in AX.

NEG AX Suppose AX =00A3 H = 0000 0000 1010 0011

Then it's 2's complement will be,

0000 0000 1010 0011 = 1111 1111 0101 1100 ← 1's complement of 00A3 H

$$\begin{array}{rl} & 1 \\ + & \overline{1111\ 1111\ 0101\ 1101} \end{array}$$ ← 2's complement of number

= FF5D H ← contents of AX after execution

7. CMP – Compare byte or word

Mnemonic CMP destination, source. **Flags** AF, OF, SF, ZF and PF, CF are updated according to the result

Algorithm Destination – source

Addr. Mode Register addressing mode

Operation

- This instruction compares a word/byte from source with byte/word from destination. The comparison is done by subtracting the source byte or word from the destination byte or word.
- The result is not stored in either of the destination or source. The destination and source remain unchanged, only flags are updated.
- The source may be register, memory location or an immediate number.
- The destination may be a register or memory location.

Compare	CF	ZF	SF	
Source > destination	1	0	1	Subtraction required borrow, so CF = 1
Source < destination	0	0	0	No borrow required CF = 0
Source = destination	0	1	0	Result of subtraction is zero

64

Example
CMP BH, CL Compares a byte in CL with byte in BH.
Let CL → 05 H and BH → 04 H, CF → 0, ZF → 0, SF → 0:

```
                        =  0000 0100  → BH
                        -  0000 0101  → CL
Carry flag → [0]           0000 0100  → Result of subtraction
Carry flag → [1]           1111 1011  ← 2's complement of the
                                        result i.e. result of
                                        comparison of BH and
                                        CL registers
```

Note : (i) The source and destination both cannot be memory locations.
(ii) The compare instructions are often used with conditional Jump instructions.
(iii) One of the operand must be in register.

12.7.3 Multiplication

Multiplication
- **MUL** (Multiply byte or word unsigned)
- **IMUL** (Multiply byte or word signed)
- **AAM** (Integer multiply byte or word ASCII adjust for multiply)

1. MUL – Multiply byte or word unsigned

Mnemonic MUL source **Flags** (i) AF, PF, SF, ZF are undefined. (ii) CF and OF will both be 0.

Algorithm When operand is a byte AX = AL * operand
When operand is a word (DX : AX) = AX * operand

Addr. Mode Register addressing mode

Operation
- This instruction multiplies an unsigned byte from source with an unsigned byte in the AL register or an unsigned word from source with an unsigned word in AX.
- When a byte is multiplied by contents of AL, the result is stored in AX. The MSB of result is stored in AH register and the LSB of result is stored in the AL register.
- When a word is multiplied by contents of AX, the product can be double word . The MSB of multiplication is stored in DX and LSB in AX register.
- The source can be a register or memory location.

- This instruction cannot be used to multiply immediate data. If we want to multiply immediate data. Then that immediate data has to be stored into some valid register and then multiplied with byte or word in AL or AX

Example
MUL CX

DX:AX =CX * AX : Result of multiplication MSB will be stored in DX register and the LSB will be stored in AX register.

2. IMUL – Multiply byte or word signed

Mnemonic IMUL source **Flags** AF, PF, SF and ZF are undefined.

Algorithm When operand is a byte → AX = AL * operand
When operand is a word → (DS : AX) = AX * operand

Addr. Mode Register Addressing mode

Operation Byte operands → AX ← AL * source
Word operands → (DS : AX) ← AX * source

- This instruction multiplies a signed byte from some source and a signed byte in AL, or a signed word from some source and a signed word in AX.
- The source can be register or memory location.
- When a signed byte is multiplied by AL a signed result will be put in AX.
- When a signed word is multiplied by AX, the MSB 16-bits are put in DX and LSB 16-bits are put in AX.
- If the magnitude of product does not require all bits of the destination, the unused bits are filled with copies of the sign bit .
- To multiply a signed byte by a signed word it is necessary to move signed byte into lower byte of word and fill the upper byte of word with copies of sign bit. This can be done by CBW instruction.
- If the upper byte of 16 bit result or upper word of 32 bit result contains only copies of sign bit (all 0's or all 1's) then the CF and OF will both be zeros.
- If the upper byte of 16 bit result or upper word of 32 bit result contains part of the product then the CF and OF will both be zeros.

Example
IMUL BL

Let AL = 69 decimal = 0100 0101 = 45H
BL = 14 decimal = 00001110 = 0EH

```
IMUL BL       45 H    ← Contents of AL register
          ×   0E H    ← Contents of BL register
          ----------
            03CE H    ← Result of multiplication  AX = 03CE H
```

MSB = 0, Positive result magnitude in true form. ∴ SF = 0, CF = OF = 1

3. AAM – Integer multiply byte or word ASCII adjust for multiply

Mnemonic AAM

Flags It updates PF, SF and ZF. The AF, CF and OF are left undefined.

Algorithm AH = Quotient of AL/10 AL = remainder of AL/10

Addr. Mode Implied addressing mode

Operation

- Numerical data coming into a computer from a terminal through keyboard is usually in ASCII code. The numbers 0 to 9 are represented by ASCII codes 30 H to 39 H.
- Before multiplying two ASCII digits, the upper nibble bits of each need to be masked. This leaves unpacked BCD in each byte. After the two unpacked BCD digits are multiplied, the AAM instruction is used to adjust the product of two unpacked BCD digits in AX.
- It works only on register AL.
- It is used after multiplying the two unpacked BCD numbers

Example

Let AL → 0000 0101 = unpacked BCD 5

BH → 0000 1001 = unpacked BCD 9

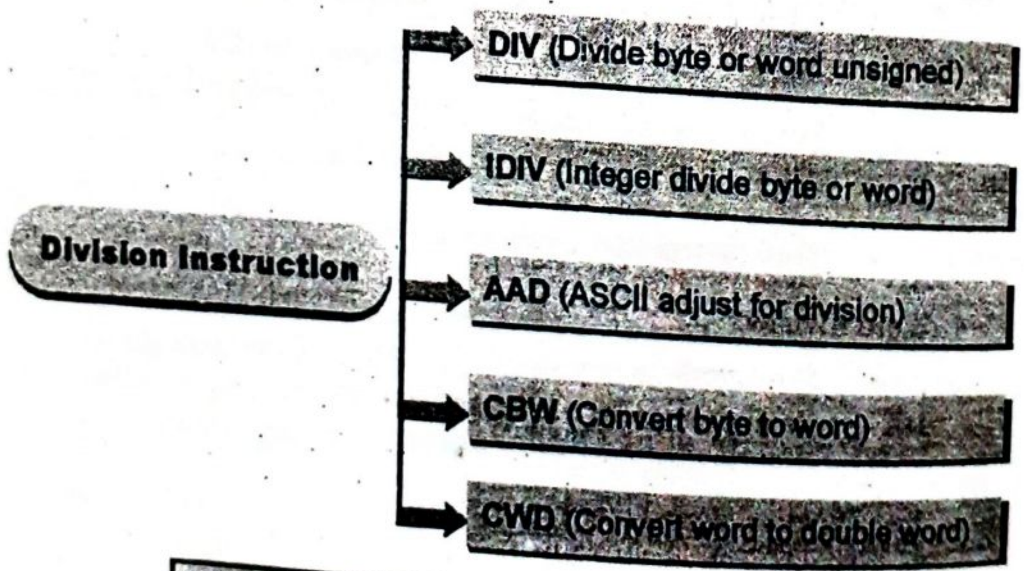
MUL BH : AL * BH → Result in AX register

AX = 0000 0000 00101101 = 002D H

AAM AX : 0000 0100 0000 0101 = 0405 H

Which is unpacked BCD for 45.

12.7.4 Division Instruction


Division Instruction
DIV (Divide byte or word unsigned)
IDIV (Integer divide byte or word)
AAD (ASCII adjust for division)
CBW (Convert byte to word)
CWD (Convert word to double word)

1. DIV – Divide byte or word unsigned

Mnemonic DIV source

Flags All flags are undefined.

Let AX = 37D7 H = 14,295 decimal,

BH = 97 H = 151 decimal

Algorithm When operand is a byte : AH = remainder (Modulus)
AL = AX/operand (Quotient) AX = (DS : AX) / operand (Quotient)
When operand is a word :
DX = remainder (modulus)

Addr. Mode Register Addressing mode

Operation This instruction basically performs two operations :
1. Divide an unsigned word by a byte.
2. Divide unsigned double word by a word.
For 1st operation, the word must be in AX register. After division,
AL → 8 bit quotient, AH → 8 bit remainder
- If an attempt is made to divide by a or quotient is too large to fit in AL. 8086 will execute a type 0 interrupt.
- For 2nd operation, MSB of word of the double word must be in DX and LSB must be in AX.
- AX → 16 bit quotient, DX → 16 bit remainder

2. IDIV – Integer divide byte or word

Flags All flags are undefined

Mnemonic IDIV source

Algorithm When operand is byte : AH = remainder (modulus)
AL = AX/operand
When operand is a word : DX = remainder (modulus)
AX = (DX : AX) / operand

Addr. Mode Register Addressing mode

Operation Byte division :
AL ← quotient of (DS : AX) / source
AH ← remainder of (DS : AX) / source
Word division :
AX ← quotient of (DS : AX) / source
DX ← remainder of (DS : AX) / source
This instruction performs two operation :
1. Divide a signed word by a signed byte
2. Divide a signed double word by a signed word.

Example A signed word divided by a single byte
Let AX = 03 AB H BL = 00 D3 H
IDIV BL
Quotient in AL = EC H Remainder in AH = 27 H

3. AAD - ASCII adjust for division

Mnemonic AAD **Flags** (i) The PF, SF and ZF are affected.
(ii) AF, CF and OF are undefined after AAD.

Algorithm AL = (AH * 10) + AL AH = 0
AL ← 10* (AH) + AL AH ← 0

Addr. Mode Implied Addressing mode

Operation
- It converts two unpacked BCD digits in AH and AL to the equivalent binary number in AL. This adjustment must be made before dividing the two unpacked BCD digits in AX, by an unpacked BCD byte. After the division,

  AL → unpacked BCD quotient

  AH → unpacked BCD remainder

Example

AX = 0607H unpacked BCD for 67 decimal	This instruction will perform following operation.
CL = 09 H, now adjust to binary using AAD.	$(AH)^* 10 = 06 * 10 = (60)_{decimal} = 3CH$
	$(AH) * 10 + (AL) = (60)_{10} + (7)_{10}$
AAD	= 3 C H + 7 H = 43 H
DIV CL	AL = 43 H and AH = 00 H
	∴ AX (New) = 0043 H
	Divide AX by unpacked BCD in CL
	Quotient AL = 07 H unpacked BCD
	Remainder AH = 04 H unpacked BCD

4. CBW - Convert byte to word

Mnemonic CBW **Flags** No flags are affected.

Algorithm If MSB bit of AL = 1 then AH = 255 (FF H)
else AH = 0

Addr. Mode Implied Addressing mode

Operation
- This instruction copies the sign bit in AL to all the bits in AH. AH is then said to be a sign extension of AL.
- This operation must be done before a signed byte in AL can be divided by another signed byte with IDIV instruction.

Example

AX = 00AC H	CBW
= – 163 decimal	Convert signed byte in AL to signed word in AX.
	Result : 1111 1111 1010 0011
	= – 163 decimal