



CSC405 Microprocessor



8086 Microprocessor

Instruction Set

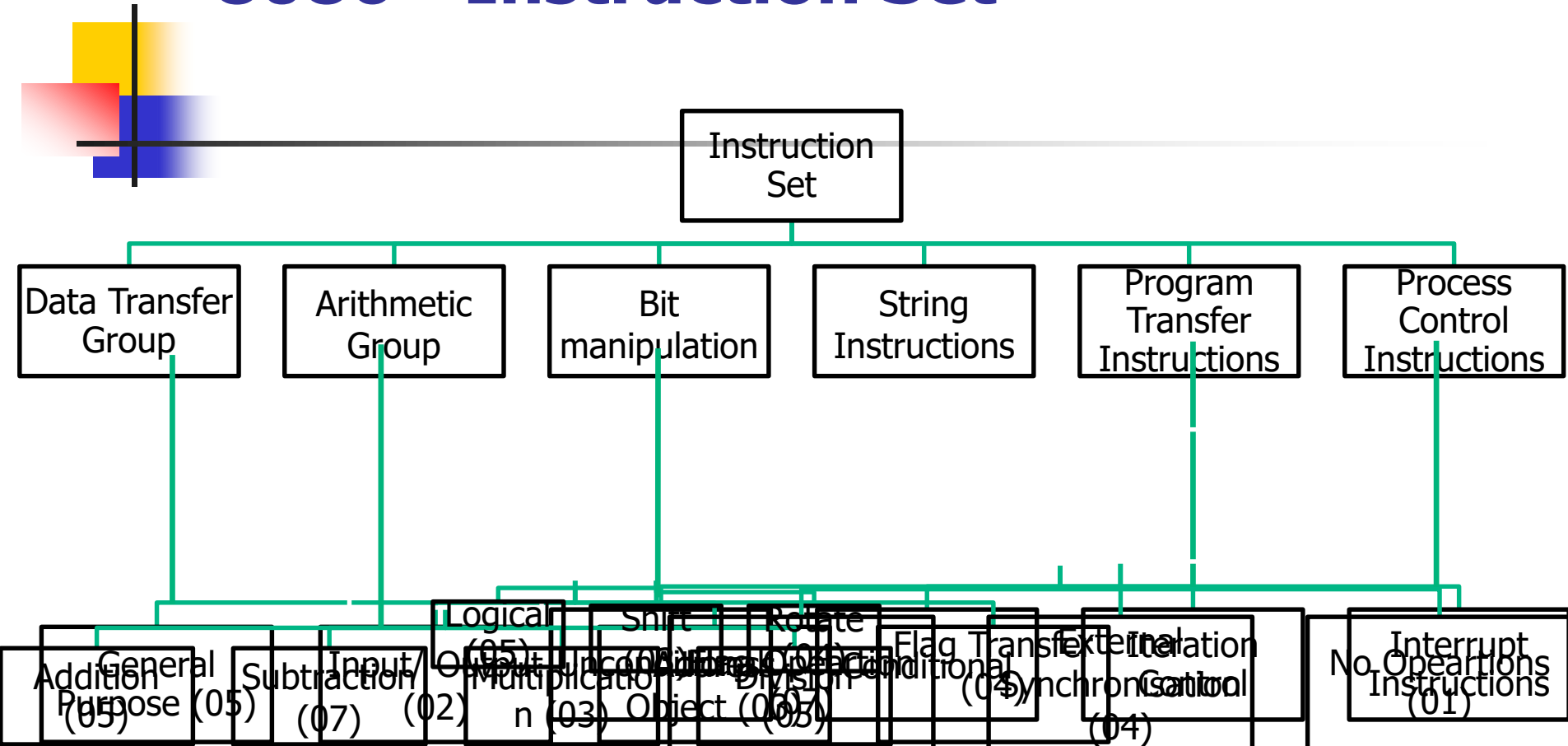


8086 - Instruction Set

Classification of Instruction Set

- 1. Data Transfer Instructions**
- 2. Arithmetic Instructions**
- 3. Bit Manipulation Instructions**
- 4. String Instructions**
- 5. Program Execution Transfer Instructions**
- 6. Processor Control Instructions**

8086 - Instruction Set





8086 - Instruction Set

Data Transfer Instructions

- These instructions are used to transfer data from source to destination.
- The operand can be
 - a constant,
 - memory location,
 - register or
 - I/O port address.



8086 - Instruction Set – Data Transfer Instructions

General Purpose Byte or Word Transfer Instruction

1. MOV : copy a Word or a Byte

MOV **destination**, **source**

MOV **operand 1**, **operand 2**

destination ← source

operand 1 ← operand 2

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

Sr. No.	Destination	Source
6	Register	Immediate
7	Memory	Immediate
8	Seg – Reg	Reg – 16
9	Seg – Reg	Mem - 16
10	Reg – 16	Seg – Reg
11	Mem - 16	Seg – Reg

8086 - Instruction Set – Data Transfer Instructions



General Purpose Byte or Word Transfer Instruction

- **MOV AX,BX**
- **MOV AH, BL**
- **MOV AX, MEMWDS**
- **MOV AL, MEMBDS**
- **MOV MEMWDS, BX**
- **MOV MEMBDS, AL**
- **MOV MEMWDS,1234H**
- **MOV MEMBDS,34H**

8086 - Instruction Set – Data Transfer Instructions



General Purpose Byte or Word Transfer Instruction

- **MOV AL,10H**
- **MOV AX,1000H**
- **MOV DS,AX**
- **MOV DX,ES**
- **MOV ES,MEMWDS**
- **MOV MEMWDS,CS**

8086 - Instruction Set – Data Transfer Instructions



2. XCHG Des, Src : Exchange byte or word

- This instruction exchanges Src with Des.
- **Src**: Register, Memory Location
- **Des**: Register, Memory Location
- It **cannot** exchange two memory locations directly.
- **E.g.:** XCHG DX, AX ; $DX \longleftrightarrow AX$

Segment reg. cannot be used as a reg. in this instruction.

8086 - Instruction Set – Special Data Transfer Instructions



3. XLAT: Translate or Replace a byte

- XLAT/XLATB
- $AL = DS:[BX + \text{unsigned } AL]$
- This instruction replaces a byte in AL register with a byte from a look up table in the memory i.e., it copies the value of memory byte at location DS : [BX + unsigned AL]
- Here the contents of AL acts as index to the desired location in lookup table

10

Segment reg. cannot be used as a reg. in this instruction.

8086 - Instruction Set – Special Data Transfer Instructions



XLAT

Translate byte in AL reg with a byte from a lookup table in memory.

- BX stores the offset of the starting address of the lookup table
- AL reg stores the byte no. from the lookup table.
- This instruction copies byte from address pointed by [BX+AL] back into AL or Move into AL the contents of memory location in Data Segment, whose offset address is formed by [BX+AL]

11

8086 - Instruction Set – Special Data Transfer Instructions



4. Input/ Output

- These instructions are basically related to communication with I/O devices.
- 2 instructions: IN & OUT

8086 - Instruction Set – Special Data Transfer Instructions

4.1. IN Accumulator, Port Address

- It transfers the operand from specified port to accumulator register.
- **E.g.:** IN AL, 28 H; AL \longleftarrow [28H]_{I/O}

4.2. OUT Port Address, Accumulator:

- It transfers the operand from accumulator to specified port.
- **E.g.:** OUT 28 H, AL; [28H]_{I/O} \longleftarrow AL

Port can be a 8 bit or 16 bit port.

8086 - Instruction Set – Special Data Transfer Instructions



5. Address Object

- These instructions manipulate the **addresses** of the variables, **rather than the contents** or values of the variables.
- These are mainly used for list processing, based variables and string operation.
- 3 types:
 - **LEA**: Load Effective Address
 - **LDS**: Load pointer using DS
 - **LES**: Load pointer using ES.

8086 - Instruction Set – Special Data Transfer Instructions



5.1 LEA Register, Src

- Load Effective Address (offset addr.)
- It **loads** a 16-bit register with the **offset address** of the data specified by the Src.
- The Reg, could be an index or a base pointer reg. – SI,DI,BX,BP
- **E.g.:** LEA BX, MEMBDS
 - Assume the address 1000H in data segment has been given label MEMBDS

8086 - Instruction Set – Special Data Transfer Instructions

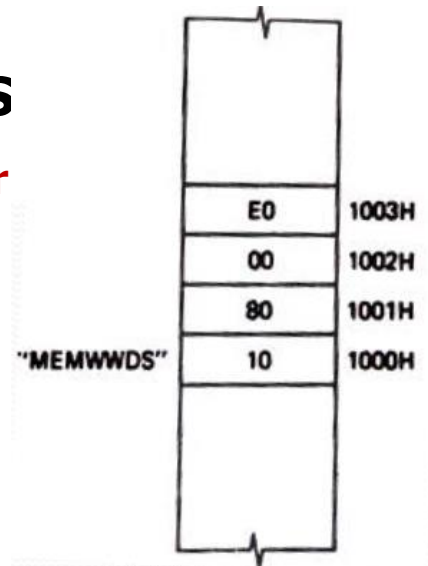
5.2. LDS reg, Src: Load pointer using DS

- The **source is always memory location** & **DS is used as segment reg** for memory.
- It is a **2 byte** instruction.
 - i. It copies a **word from 2 memory locations** into **reg specified in the instruction**.
 - ii. It copies a word from the **next 2 memory locations** into **DS reg**.
- It loads **32-bit pointer** from memory source to **destination register** and **DS**.

8086 - Instruction Set – Special Data Transfer Instructions

5.2. LDS dest, Src : Load pointer using DS

- The **offset addr.** is placed in the **destination register** and the **segment addr.** is placed in **DS**.
- To use this instruction the **word at the lower memory address must contain the offset addr** and the word at the higher address must contain the segment addr.



MEMWWDS defines a double word beginning at address 1000H

- **E.g.:** LDS BX, MEMWWDS

BX ← {DS: [MEMWWDS], DS:[MEMWWDS +1]}

DS ← {DS:[MEMWWDS +2], DS:[MEMWWDS +3]}

8086 - Instruction Set – Special Data Transfer Instructions



5.3. LES reg, Src: Load pointer using ES

- The source is always memory location.
- It is a 2 byte instruction.
 - i. It copies a **word from 2 memory locations** into reg specified in the instruction.
 - ii. It copies a word from the **next 2 memory locations** into ES reg
- It loads **32-bit pointer** from memory source to **destination register** and **ES**.

8086 - Instruction Set – Special Data Transfer Instructions

5.3. LES reg, Src: Load pointer using ES

- The **offset addr.** is placed in the **destination register** and the **segment addr** is placed in **ES**.
- This instruction is very similar to LDS except that **it initializes ES** instead of DS.
- **E.g.:** LES BX, MEMWWDS

BX ← {DS: [MEMWWDS], DS:[MEMWWDS +1]}

ES ← {DS:[MEMWWDS +2], DS:[MEMWWDS +3]}

8086 - Instruction Set – Special Data Transfer Instructions

5.3. LES reg, Src: Load pointer using ES

- The **offset addr.** is placed in the **destination register** and the **segment addr** is placed in **ES**.
- This instruction is very similar to LDS except that **it initializes ES** instead of DS.
- **E.g.:** LES BX, MEMWWDS

BX ← {DS: [MEMWWDS], DS:[MEMWWDS +1]}

ES ← {DS:[MEMWWDS +2], DS:[MEMWWDS +3]}

8086 - Instruction Set – Special Data Transfer Instructions



6. Stack memory related instruction:

6.1 PUSH source

6.2 POP destination

8086 - Instruction Set – Special Data Transfer Instructions



6.1. PUSH source

Source: Register or Memory location

Destination: Top of Stack

- **Push** 16-bit source data into the top of stack.
- Know that SP always points to the top of the stack.
- Since stack memory is byte organized, pushing 16-bit data, will require 2 locations.
- Note that Stack grows downwards.
- So, after one execution of PUSH instruction SP is decremented by 2.

8086 - Instruction Set – Special Data Transfer Instructions

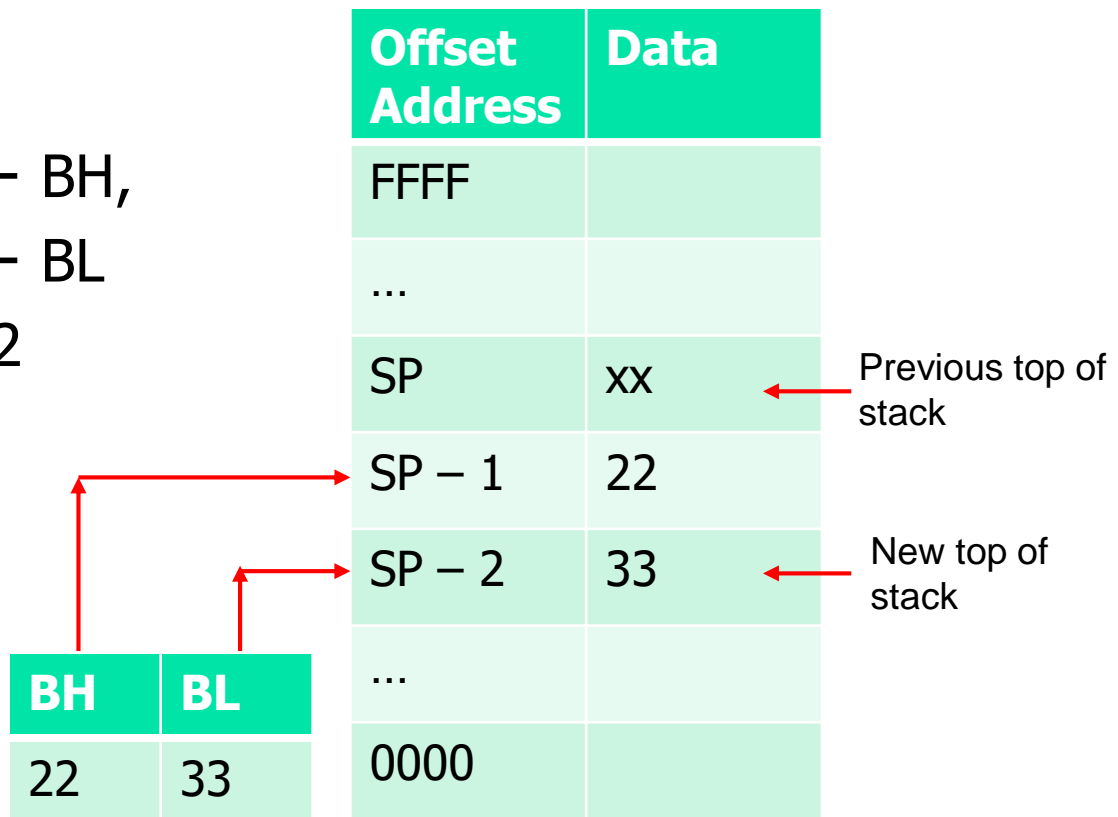
Example:

PUSH BX

$SS:[SP-1] \leftarrow BH,$

$SS:[SP-2] \leftarrow BL$

$SP \leftarrow SP - 2$



8086 - Instruction Set – Special Data Transfer Instructions



6.2. POP destination

Source: Top of Stack

Destination: Register (except CS) or Memory location

- POP 16-bit source data from the top of stack and store into the destination.
- Know that SP always points to the top of the stack.
- So, after one execution of POP instruction SP is incremented by 2.

8086 - Instruction Set – Special Data Transfer Instructions

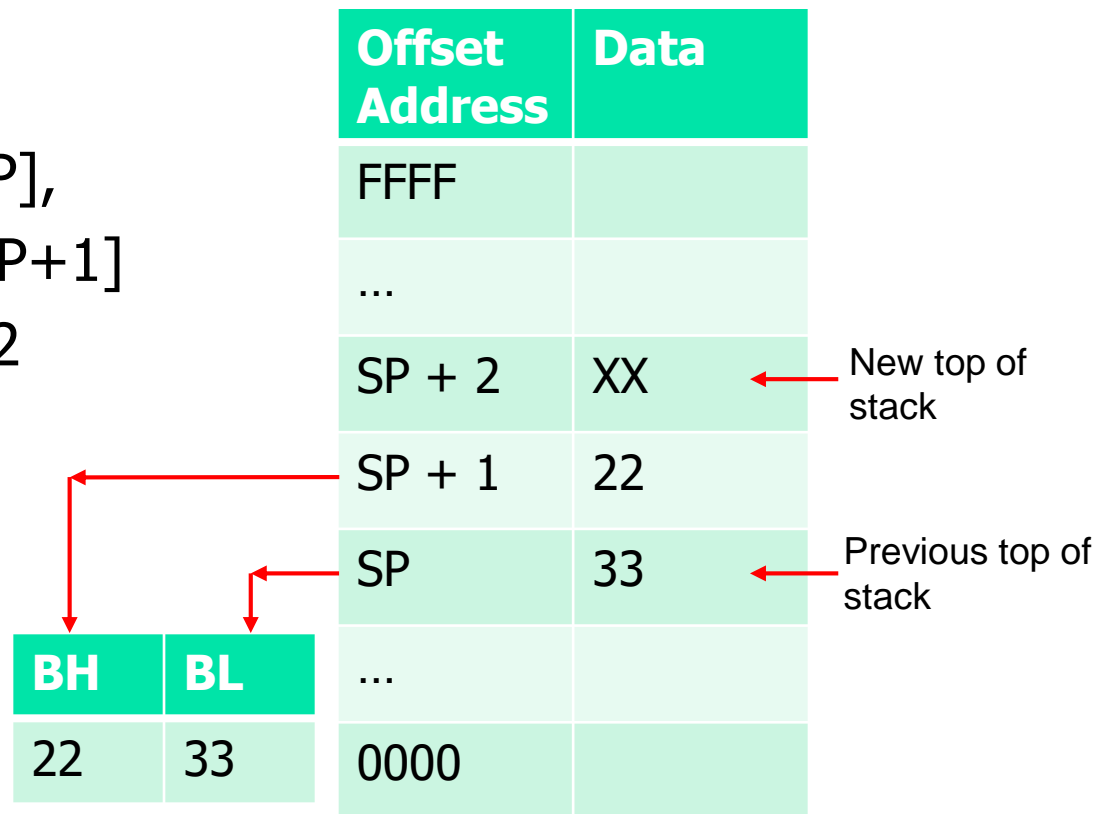
Example:

POP BX

$BL \leftarrow SS:[SP],$

$BH \leftarrow SS:[SP+1]$

$SP \leftarrow SP + 2$



8086 - Instruction Set – Special Data Transfer Instructions



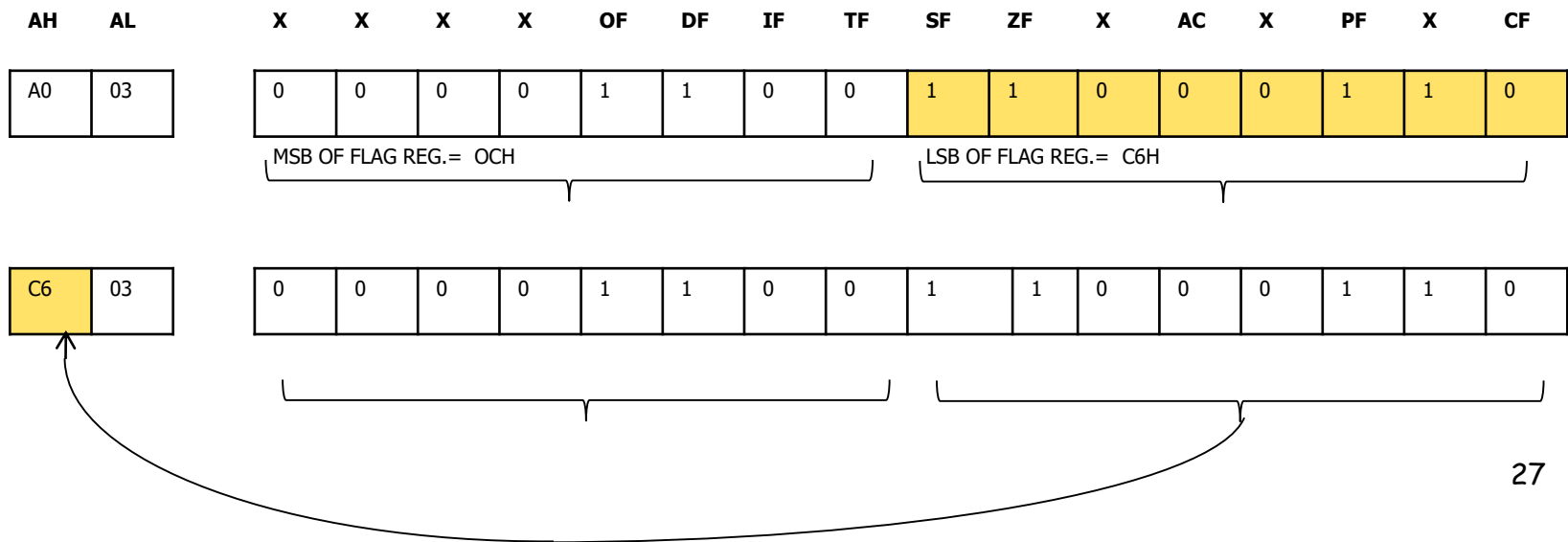
7. Flag Instructions

- These instructions are related to movement of flag register to/from a register & memory
- 4 instructions:
 - **LAHF**
 - **SAHF**
 - **PUSHF**
 - **POPF**

8086 - Instruction Set – Special Data Transfer Instructions

7.1. LAHF

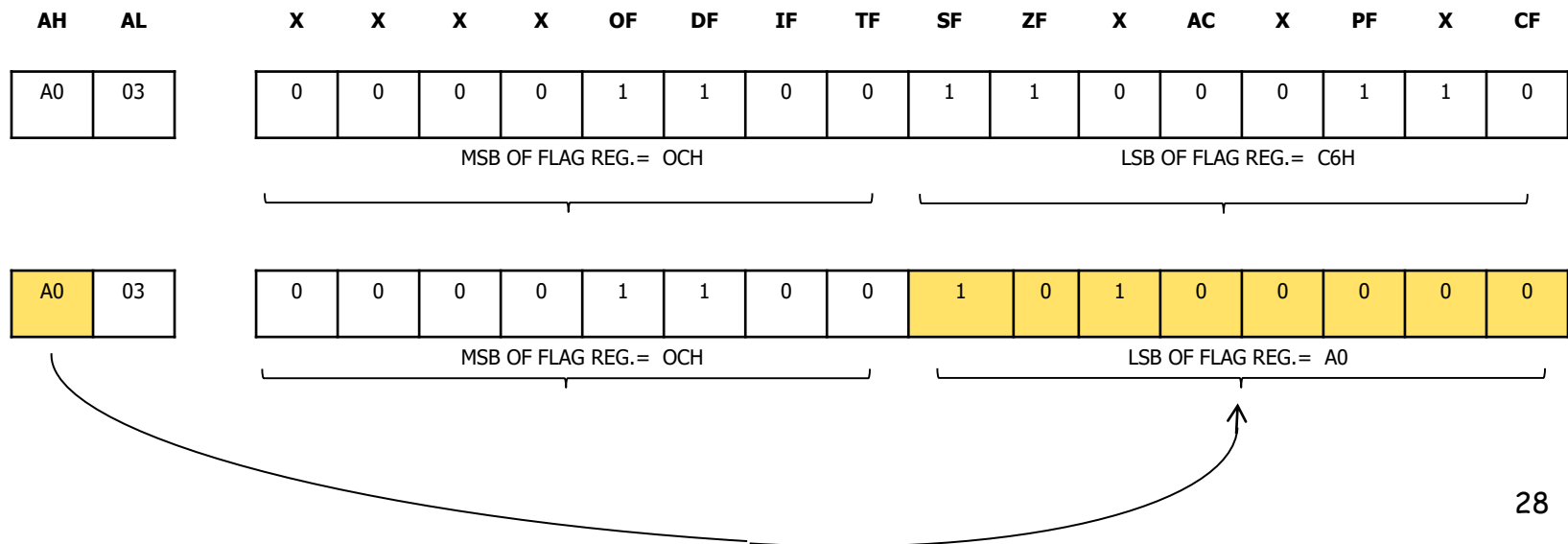
- The lower byte of flag register is copied to the AH reg.



8086 - Instruction Set – Special Data Transfer Instructions

7.2. SAHF

- It stores the contents of AH to lower byte of flag register.



8086 - Instruction Set – Special Data Transfer Instructions



7.3 PUSHF

Push the value of Flag register into stack and decrement the stack pointer by 2.

$$SS:[SP-1] \leftarrow FLAG_H,$$
$$SS:[SP-2] \leftarrow FLAG_L$$
$$SP \leftarrow SP - 2$$

8086 - Instruction Set – Special Data Transfer Instructions



7.4 POPF

Pop a word from the stack into the Flag register.

$$\text{FLAG}_L \leftarrow \text{SS}:[\text{SP}],$$
$$\text{FLAG}_H \leftarrow \text{SS}:[\text{SP}+1]$$
$$\text{SP} \leftarrow \text{SP} + 2$$



8086 - Instruction Set

Arithmetic Instructions

Addition	
ADD	: Add byte or word
ADC	: Add byte or word with carry
INC	: Increment byte or word by 1
AAA	: ASCII adjust for addition
DAA	: Decimal adjustment for addition



8086 - Instruction Set

Arithmetic Instructions

Subtraction	
SUB	: Subtract byte or word
SBB	: Subtract byte or word with borrow
DEC	: Decrement byte or word by1
NEG	: Negate byte or word
CMP	: Compare byte or word
AAS	: ASCII adjust for subtraction
DAS	: Decimal adjustment for subtraction



8086 - Instruction Set

Arithmetic Instructions

Multiplication	
MUL	: Multiply byte or word unsigned
IMUL	: Multiply byte or word signed/ Integer Multiply byte or word
AAM	: ASCII adjust for multiply



8086 - Instruction Set

Arithmetic Instructions

Division	
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

8086 - Instruction Set – Arithmetic Instructions

I. Addition Instructions

1) ADD Des, Src:

- It adds a byte to byte or a word to word.
- Adds a number from src to des and the result is in des.
- **Src**: Register, Memory Location or Immediate number.
- **Des**: Register, Memory Location.
- **Flags affected**: AF, CF, OF, PF, SF, ZF

E.g.: ADD AL, 74H

 ADD DL, CL

 ADD DX, AX

 ADD BYTE PTR [BX], CH

8086 - Instruction Set – Arithmetic Instructions

2) ADC Des, Src:

- It adds the two operands **with CF**.
- **Src**: Register, Memory Location or immediate number.
- **Des**: Register, Memory Location.
- **Flags affected**: AF, CF, OF, PF, SF, ZF

E.g.: ADC AL, 74H
 ADC DX, AX
 ADC BYTE PTR [BX], CH

8086 - Instruction Set – Arithmetic Instructions

32 bit Addition

BX:AX
+ DX:CX

DX:CX

ADD CX, AX
ADC DX, BX

8086 - Instruction Set – Arithmetic Instructions

3) INC dest

- It increments the byte or word by one. Or add 1 to specified dest.
- **dest**: Register or Memory location.
- **Flags affected**: AF, OF, PF, SF, ZF

CF is **not** affected.

E.g.: INC AX

INC BL

INC WORD PTR[DI]

INC MEMBDS

8086 - Instruction Set – Arithmetic Instructions

II. Subtraction Instructions:

4) SUB Des, Src:

- It subtracts a byte from byte or a word from word.
- Subtracts a number in the src from des and the result is in des.
- **Src**: Register ,Memory Location or immediate number.
- **Des**: Register ,Memory Location
- **Flags affected**: AF, CF, OF, PF, SF, ZF flags.
- For subtraction, **CF acts as borrow flag**.
- **E.g.:**
SUB AL, 74H
SUB DX, AX
SUB BYTE PTR [BX], CH

8086 - Instruction Set – Arithmetic Instructions

5) SBB Des, Src:

- It subtracts the two operands and also the **borrow** from the result.
- **Src**: Register ,Memory Location or immediate number.
- **Des**: Register ,Memory Location
- **Flags affected**: AF, CF, OF, PF, SF, ZF flags.
- **E.g.:** SBB AL, 74H
SBB DX, AX
SBB BYTE PTR [BX], CH

8086 - Instruction Set – Arithmetic Instructions

6) DEC dest:

- It decrements the byte or word by one.
- **dest:** Register or Memory location.
- **Flags affected:** AF, OF, PF, SF, ZF
CF is not affected.

E.g.: DEC AX
DEC BL
DEC WORD PTR[DI]
DEC MEMBDS

8086 - Instruction Set – Arithmetic Instructions

7) NEG dest:

- It creates 2's complement of a given number and stores it back in the dest.
- That means, it changes the sign of a number.
- **dest:** Register, Memory location
- **Flags affected:** AF, OF, PF, SF, ZF,

CF = 1 EXCEPT WHEN THE OPERAND IS ZERO

- Assume AL = 0000 0101 = 05H then

NEG AL; AL \leftarrow 1111 1011 = FBH

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

8086 - Instruction Set – Arithmetic Instructions

8) CMP Des, Src:

- It compares two specified bytes or words.
- **Src:** Immediate number, register or memory location.
- **Des:** Register or memory location.
- Both operands cannot be a memory location at the same time.
- The comparison is done simply by internally **subtracting** the source from destination.
- The value of source and destination **does not** change, but the **flags are modified** to indicate the result.
- **Flags affected:** AF, OF, PF, SF, ZF, CF
- **E.g.:** CMP BL,55H
 CMP CX,BX

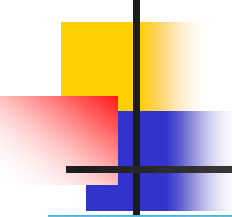
8086 - Instruction Set – Arithmetic Instructions

III. Multiplication Instruction

9) **MUL Src:** (unsigned multiplication) **8/16 bit**

- It multiplies two bytes to produce a word or two words to produce a double word.
- **$AX = AL * Src$ (8 bit)**
- **$DX : AX = AX * Src$ (16 bit)**
- This instruction assumes one of the operand in AL or AX.
- **Src:** register or memory location.
- **E.g.:** $MUL\ BL; \quad AX = AL * BL$
 $MUL\ BX; \quad DX : AX = AX * BX$

8086 - Instruction Set – Arithmetic Instructions



General mnemonic		Object code	Mnemonic	Segment for memory access	Symbolic operation	Description
Op-code	Operand*					
MUL	<i>source</i>	F6 E3	MUL BL	Within CPU	$AX \leftarrow AL * BL$	Unsigned multiplication of the byte or word source operand and the accumulator; word results are stored in AX and double word results are stored in DX:AX (see Fig. 2.14); if the result cannot be stored in a single byte (for byte multiplication) or a single word (for word multiplication) CF and OF are set; cleared otherwise, all other flags are undefined
		F7 E1	MUL CX	Within CPU	$DX:AX \leftarrow AX * CX$	
			MUL BYTE PTR[BX]	Data	$AX \leftarrow AL * [BX]$	
		F6 27 F7 26 00 10	MUL MEMWDS ^b	Data	$DX:AX \leftarrow AX * [1001H:1000H]$	

8086 - Instruction Set – Arithmetic Instructions

IV. Division

10. DIV Src: (unsigned division) 8/16 bit reg – divisor

- It divides **word by byte** or **double word by word**.
- If the divisor is 8 bit then dividend is stored in AX, and the result is stored as:
 - **AH = remainder, AL = quotient**
- If the divisor is 16 bit then dividend is stored in DX: AX, and the result is stored as: **DX = remainder AX = quotient**
- **ALL flags are undefined** after DIV instruction
- **E.g.:** DIV BL; AX / BL


8086 - Instruction Set – Arithmetic Instructions



General mnemonic		Object code	Mnemonic	Segment for memory access	Symbolic operation	Description
Op-code	Operand ^a					
DIV	<i>source</i>	F6 F3 F7 F1 F6 37 F7 36 00 10	DIV BL DIV CX DIV BYTE PTR[BX] DIV MEMWDS ^b	Within CPU Within CPU Data Data	AX ← AX/BL DX:AX ← AX/CX AX ← AX/[BX] DX:AX ← AX/[1001H:1000H]	Unsigned division of the accumulator (for byte divisors) or accumulator and DX (for word divisors); for byte divisors the result is returned in AL with the remainder in AH; for word divisors the result is returned in AX with remainder in DX (see Fig. 2.14); if the quotient exceeds the capacity of its destination register (AL or AX) a type 0 interrupt is generated; all flags are undefined


Ex: DIVIDING 65,000 BY 2

8086 - Instruction Set – Arithmetic Instructions



General mnemonic		Object code	Mnemonic	Segment for memory access	Symbolic operation	Description
Op-code	Operand ^a					
IMUL	<i>source</i>	F6 EB	IMUL BL	Within CPU	$AX \leftarrow AL * BL$ (signed)	Same as MUL except signed numbers are allowed; the source operand is limited to -128 to +127 for byte multiplication and -32768 to +32767 for word multiplication; CF and OF are set if the result cannot be represented in the low order register of the result; cleared otherwise, the sign is extended to the high order register; the other flags are not affected
		F7 E9	IMUL CX	Within CPU	$DX:AX \leftarrow AX * CX$ (signed)	
		F6 2F	IMUL BYTE PTR[BX]	Data	$AX \leftarrow AL * [BX]$ (signed)	
		F7 2E 00 10	IMUL MEMWDS ^b	Data	$DX:AX \leftarrow AX * [1001H:1000H]$ (signed)	

8086 - Instruction Set – Arithmetic Instructions



General mnemonic		Object code	Mnemonic	Segment for memory access	Symbolic operation	Description
Op-code	Operand ^a					
IDIV	<i>source</i>	F6 FB F7 F9 F6 3F F7 3E 00 10	IDIV BL IDIV CX IDIV BYTE PTR[BX] IDIV MEMWDS ^b	Within CPU Within CPU Data Data	$AX \leftarrow AL/BL$ (signed) $DX:AX \leftarrow AX/CX$ (signed) $AX \leftarrow AL/[BX]$ (signed) $DX:AX \leftarrow AX/[1001H:1000H]$ (signed)	Same as DIV except signed division is performed; the source operand is limited to -128 to +127 for byte division and -32768 to +32767 for word division

8086 - Instruction Set – Arithmetic Adjust Instruction

- **BCD** – 4 bits to represent the digit 0-9
- Packed decimal and Unpacked decimal
- **ASCII** – similar to unpacked decimal (one byte can hold one digit)
- To convert ASCII to decimal – simply subtract 30H
- Consider the following addition of two unpacked decimal numbers.

MOV AL,7;

MOV BL, 5;

ADD AL, BL;

AL=0CH **not** 12.

So, how to get 12(decimal) as answer?

8086 - Instruction Set – Arithmetic Adjust Instruction

13. DAA (Decimal Adjust after Addition)

- It is used to make sure that the result of adding two BCD numbers is adjusted to be a correct BCD number.
- It only works on AL register.
- **Flags** updated: **All** flags affected **except OF** (**OF is undefined**)

8086 - Instruction Set – Arithmetic Adjust Instruction

DAA (Decimal Adjust after Addition)

- If the value of low order 4 bits (D0-D3) in the AL reg is greater than 9 or if AF is set, the instruction adds 06H to the result
 - If lower nibble of AL > 9 or AF=1
then $AL = AL + 06$, AF=1
- If the value of higher order 4 bits (D4-D7) in the AL reg is greater than 9 or if CF is set, the instruction adds 60 to the result
 - If higher nibble of AL > 09 or CF=1
then $AL = AL + 60H$, CF=1

8086 - Instruction Set – Arithmetic Adjust Instruction



MOV AL,7;

MOV BL, 5;

ADD AL, BL; AL = 0CH

DAA; AL = 12H

AL = 7 + 5 = 0CH and not 12!!

Use DAA –Decimal Adjust for Addition

Now AL = 12H

8086 - Instruction Set – Arithmetic Adjust Instruction

ADD AL,BL

DAA

AL = 10h

AL = 26

AL = 39

Al = 39

BL = 20h

BL = 53

BL = 26

BL = 28

AL = 70

AL = 80

AL = 99

BL = 60

BL = 90

BL = 99

8086 - Instruction Set – Arithmetic Adjust Instruction

14. DAS (Decimal Adjust after Subtraction)

- It is used to make sure that the result of subtracting two BCD numbers is adjusted to be a correct BCD number.
- It only works on AL register.
- **Flags updated:** AF, CF, PF & ZF. OF is undefined
- If the value of low order 4 bits (D0-D3) in the AL reg is greater than 9 or if AF is set, the instruction subtracts 06 to the result
- If the value of higher order 4 bits (D4-D7) in the AL reg is greater than 9 or if CF is set, the instruction subtracts 60 to the result

55

8086 - Instruction Set – Arithmetic Adjust Instruction

15. AAA (ASCII Adjust after Addition): (unpacked decimal numbers only)

- Executed after addition of 2 ASCII data to convert the result in AL to correct unpacked BCD form.
- AAA instruction works only on AL reg.
- If **lower nibble** of AL (i.e., AL.0F) > 9 or **AF=1**

then, $AL = AL + 6$

$AH = AH + 1$

$AF \leftarrow 1; CF \leftarrow 1$

$AL \leftarrow AL.0F$

Ex: MOV AH,0;
MOV AL,7;
MOV BL, 5;
ADD AL, BL;
AAA;
ADD AX, 3030H;

AH = 0000 0000

AL = 0000 0111

BL = 0000 0101

AL = 0000 1100 (0CH)

AX = 0102H ⁵⁶

AX = 3132H

8086 - Instruction Set – Arithmetic Adjust Instruction

16. AAS (ASCII Adjust after Subtraction):

- Similar like AAA
- Its result is obtained in unpacked BCD form.
- This instruction does not have any operand.
- **Flag Affected** : AF & CF. But OF, PF, SF, ZF are undefined

8086 - Instruction Set – Arithmetic Adjust Instruction

17. AAM (ASCII Adjust for Multiplication)

CX = 3639H; Multiply two ASCII digits in CH and CL. Store ASCII result in AH and AL.

AND CX, 0F0FH; **unpack** CH and CL => CX=0609H

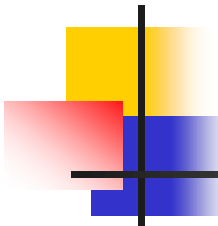
MOV AL, CH; multiplier to AL

MUL CL; AX = 00**36**H

AAM; form two unpack decimal digits AX = **0504**H

ADD AX, 3030H; AX = 3534H

AAM instruction can be considered a special form of the DIV instruction. This is because it divides AL by 10D, leaving the quotient in AH and the remainder in AL.



General mnemonic		Object code	Coding example			
			Mnemonic	Segment for memory access	Symbolic operation	Description
AAM	none	D4 0A	AAM	Within CPU	$AH \leftarrow AL/0AH$ $AL \leftarrow \text{remainder}$	Following the multiplication of two valid unpacked decimal operands, AAM converts the result in AL to two valid unpacked decimal digits in AH and AL; all flags except PF, SF, and ZF are undefined are undefined
AAD	none	D5 0A	AAD	Within CPU	$AL \leftarrow (AH * 0AH) + AL$ $AH \leftarrow 0$	Before dividing AX by a single-digit unpacked decimal operand, AAD converts the two-digit unpacked decimal number in AX to a binary number in AL and 0 in AH; the quotient produced by the following division will then be a valid unpacked decimal number in AL and remainder in AH; all flags except PF, SF, and ZF are undefined

8086 - Instruction Set – Arithmetic Adjust Instruction

18. AAD (ASCII Adjust for Division)

- AAD should be used **before** the division. (2 digit unpacked decimal no divide by decimal operand)
- This will convert AX to a binary number in AL and 00 in AH.
- The result of division will be two unpacked decimal numbers –
 - quotient in AL and remainder in AH.

AX = 0607H (67D), CL = 09H.

AAD; AX = 0043H (67D)

DIV CL; AL = 07H and AH = 04H

8086 - Instruction Set – Arithmetic Instructions

- CBW AND CWD (used for signed division)
- To divide two signed bytes –IDIV requires one of the number to be in AX

CBW converts a **byte** in **AL** to a **word** in **AX**

If AL = FBH; (-5D), after CBW, AX = FFFBH;

CWD performs the same function to divide two 16-bit words

CWD converts a word in AX to a double word in DX:AX

8086 - Instruction Set – Arithmetic Instructions

19. CBW (Convert signed Byte to signed Word):

- This instruction converts byte in AL to word in AX
- This instruction copies the sign of the byte in AL to all bits in AH
- AH is said to be sign extension of AL
- **Ex:** Assume AX = XXXX XXXX **1001 1000**

Then CBW gives AX= **1111 1111 1001 0001**

8086 - Instruction Set – Arithmetic Instructions

20. CWD (Convert Word to Double Word):

- This instruction converts word in AX to double word in DX : AX.
- It copies sign of the word in AX into all the bits of DX.
- DX is then called sign extension of AX
- The conversion is done by extending the sign bit of AX throughout DX.
- **Ex:** Assume AX = **1000 0000 1001 0001**

DX = xxxx xxxx xxxx xxxx

- Then CWD gives AX = **1000 0000 1001 0001**

DX = **1111 1111 1111 1111**

8086 - Instruction Set – Arithmetic Instructions

General mnemonic		Object code	Coding example			
Op-code	Operand		Mnemonic	Segment for memory access	Symbolic operation	Description
CBW	none	98	CBW	Within CPU	If AL < 80H, then AH ← 0 If AL > 7F, then AH ← FFH	Before dividing AX by a byte operand, CBW extends the sign of a byte dividend in AL into AH, thus converting AL into a valid signed word in AX; none of the flags are affected
CWD	none	99	CWD	Within CPU	If AX < 8000H, then DX ← 0 If AX > 7FFFH, then DX ← FFFFH	Same as CBW but extends the sign of a word dividend in AX into a double word in DX:AX; none of the flags are affected

8086 - Instruction Set – **Logical** Instructions

Bit Manipulation Instructions:

- These instructions are used at the bit level.
- These instructions can be used for:
 - i. Testing a zero bit
 - ii. Set or reset a bit
 - iii. Shift bits across registers

8086 - Instruction Set – **Logical** Instructions

Bit Manipulation Instructions

LOGICALS	
NOT	: Not byte or word
AND	: And byte or word
OR	: Inclusive OR byte or word
XOR	: Exclusive OR byte or word
TEST	: Test byte or word

8086 - Instruction Set – **Logical** **Instructions**

Bit Manipulation Instructions

SHIFTS	
SHL/SAL	: Shift logical/ arithmetic left byte or word
SHR	: Shift logical right byte or word
SAR	: Shift arithmetic right byte or word
ROTATES	
ROL	: Rotate left byte or word
ROR	: Rotate right byte or word
RCL	: Rotate through carry left byte or word
RCR	: Rotate through carry right byte or word

8086 - Instruction Set – Logical Instructions

1. NOT Src:

- It complements each bit of Src to produce 1's complement of the specified operand.
- Src: register or memory location.
- None of the flags are affected
- **Ex:** NOT BX ; BX <--- \overline{BX}
NOT BYTE PTR[SI] ; [SI] <--- $\overline{[SI]}$

8086 - Instruction Set – Logical Instructions

2. AND Des, Src:

- It performs AND operation of Des and Src.
- **Src:** Immediate number, register or memory location.
- **Des:** register or memory location.
- Both operands cannot be memory locations at the same time.
- **Flags Affected:** CF and OF are **reset**. PF, SF and ZF are updated. AF is undefined
- **Eg:** AND CX, DX ; CX <-- CX.DX
AND AX, 8000H ; AX <--- AX.8000H
AND BH, BYTE PTR[SI] ; BH <--- BH.[SI]

8086 - Instruction Set – Logical Instructions

Example:

MOV AL, 6DH;

MOV BH, 40H;

AND AL, BH;

AL – 0110 1101

BH – 0**1**00 0000

AND – 0**1**00 0000 in AL

The **flags** are affected as

CF = 0

PF = 0

AF = X

ZF = 0

SF = 0

OF = 0

8086 - Instruction Set – Logical Instructions

3. OR Des, Src:

- It performs OR operation of Des and Src.
- **Src:** immediate number, register or memory location.
- **Des:** register or memory location.
- Both operands cannot be memory locations at the same time.
- **Flags Affected:** CF and OF are **reset**. PF, SF and ZF are updated. AF is undefined
- **Eg:**
OR CX,DX ; CX <-- CX + DX
OR AX, 8000H ; AX <--- AX +8000H
OR BH, BYTE PTR[SI] ; BH <--- BH + [SI]

4. XOR Des, Src:

- It performs XOR operation of Des and Src.
- **Src:** immediate number, register or memory location.
- **Des:** register or memory location.
- Both operands cannot be memory locations at the same time.
- **Flags Affected:** CF and OF are **reset**. PF, SF and ZF are updated. AF is undefined

- **Eg:** XOR CX,DX ; CX ← CX ⊕ DX
XOR AX, 8000H ; AX ← AX ⊕ 8000H

- This instruction logically AND's the source with the destination but the result is not stored anywhere.
- Both the Src and Des should be of same size.
- Src:** immediate number, register or memory location.
- Des:** register or memory location.
- CF and OF are reset. PF, SF and ZF are updated. AF is undefined
- Eg:** TEST CX, DX ; CX.DX
 TEST AX, 8000H ; AX.8000H
 TEST BH, BYTE PTR[SI] ; BH.[SI]

8086 - Instruction Set – **Logical Instructions**

Applications

- AND operation
 - Selected bits of the destination operand can be forced low by choosing those bits as 0 in the source operand => **Masking**
 - Source operand can be taken as mask for testing selected bits of the destination operand.

AND AL, BH

TEST AL, 40H

JZ START

JZ START

- TEST instruction performs the same operation but does not alter the source/destination operands.

8086 - Instruction Set – Logical Instructions

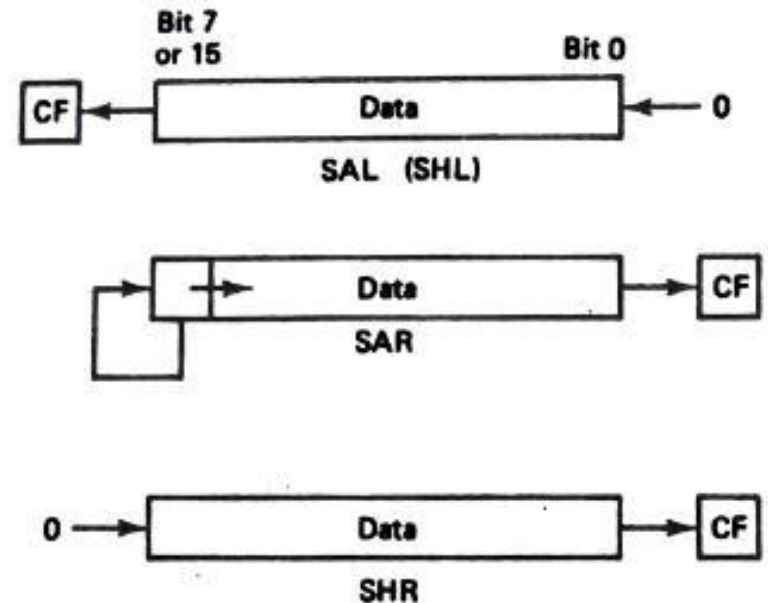
Applications

- The OR instruction can be used to **force selected** bits **high**.
 - OR AL, 80H; force bit 7 of AL high without altering other bits
- The XOR instruction can be used to **compliment selected** bits.
 - XOR AL, 80H; compliment bit 7 of AL without altering other bits

8086 - Instruction Set – Shift and Rotate Instructions

Shift Instructions

- SAL / SAR – Arithmetic
- SHL/ SHR – Logical

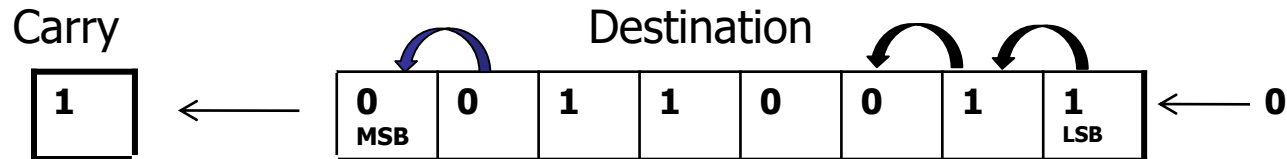


8086 - Instruction Set – Shift and Rotate Instructions

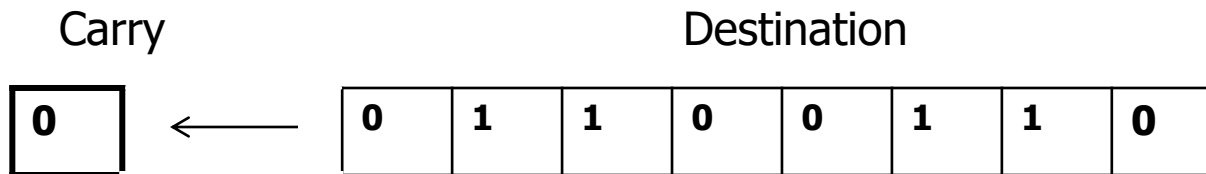
6. SAL/SHL :

Eg: If Count = 1, SAL BL,1

Assume before operation BL=0011 0011 & CF=1



After operation BL=0110 0110 & CF=0



Eg: Count >1

MOV CL, 05H

SAL BL,CL

8086 - Instruction Set – Shift and Rotate Instructions

7. SHR Des, Count:

- It shift bits of byte or word right, by count
- It puts **zero(s) in MSBs** and **LSB is shifted into carry flag**.
- **Des**: Register or Memory Location
- Bits are shifted 'count' no. of types

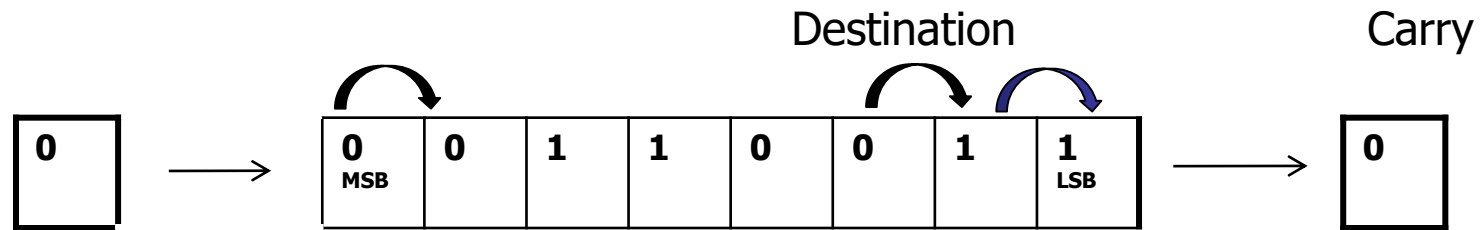
If **count = 1**, it is directly specified in the instruction as count

If **count > 1**, count is put in **CL** register & CL gives the count in the instruction.

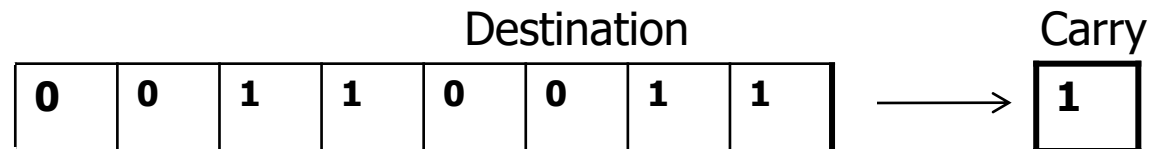
8086 - Instruction Set – Shift and Rotate Instructions

Eg: If Count = 1, SHR BL,1

Assume before operation BL=0011 0011 & CF=0



After operation BL=0001 1001 & CF= 1



Eg: Count >1

MOV CL, 05H

SHR BL,CL

8086 - Instruction Set – Shift and Rotate Instructions

8. SAR Des, Count:

- It right shifts bits of destination byte or word right, by count
- A copy of **old MSB is placed in MSB** itself and **LSB is shifted into carry flag.**
- **Des:** Register or Memory Location
- Bits are shifted 'count' no. of times

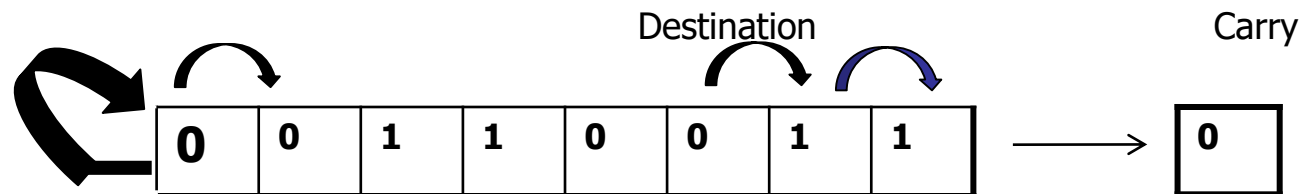
If count = 1, it is directly specified in the instruction as count

If count > 1, count is put in CL register & CL gives the count in the instruction.

8086 - Instruction Set – Shift and Rotate Instructions

Eg: Count = 1 ;SAR BL,1

Assume before operation BL=0011 0011 & CF=0



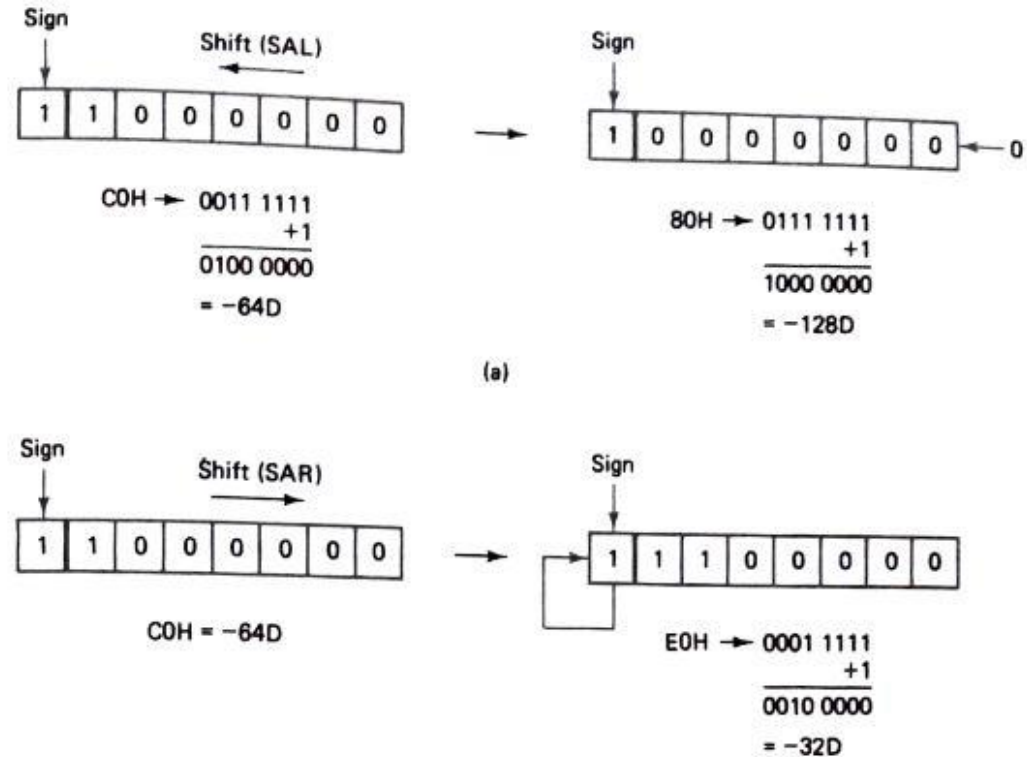
After operation BL=0001 1001 & CF=1



Eg: Count > 1
MOV CL, 05H
SAR BL,CL

8086 - Instruction Set – Shift and Rotate Instructions

- SAL – multiplies data by 2
- SAR – divides the data by 2
- OF – Set if
 - Quantity exceeds the limits for an 8- or 16-bit register
(+127 to -128 for bytes
+32767 to -32768 for words)
- AF is undefined
- All other flags are updated



8086 - Instruction Set – Shift and Rotate Instructions

General mnemonic		Coding example ^a				
Op-code	Operand	Object code	Mnemonic	Segment for memory access	Symbolic operation	Description
SAL/SHL ^b	<i>dest,count</i>	D1 E0	SAL AX,1	Within CPU	See Fig. 2.12	Shift word or byte operand left or right once or CL times; AF is undefined, all other flags are updated; for single-bit shifts, OF is set if the sign of the operand changes
		D3 E0	SAL AX,CL	Within CPU	See Fig. 2.12	
SAR	<i>dest,count</i>	D0 F8	SAR AL,1	Within CPU	See Fig. 2.12	
SHR	<i>dest,count</i>	D2 F8	SAR AL,CL	Within CPU	See Fig. 2.12	
		D1 2C	SHR WORD PTR[SI],1	Data	See Fig. 2.12	
		D2 2C	SHR BYTE PTR[SI],CL	Data	See Fig. 2.12	
RCL	<i>dest,count</i>	D1 D3	RCL BX,1	Within CPU	See Fig. 2.12	Rotate word or byte operand left or right once or CL times; only CF and OF are affected; for single-bit rotates, OF is set if the sign of the operand changes
		D3 D3	RCL BX,CL	Within CPU	See Fig. 2.12	
RCR	<i>dest,count</i>	D0 DB	RCR BL,1	Within CPU	See Fig. 2.12	
		D2 DB	RCR BL,CL	Within CPU	See Fig. 2.12	
ROL	<i>dest,count</i>	D1 04	ROL WORD PTR[SI],1	Data	See Fig. 2.12	
		D2 04	ROL BYTE PTR[SI],CL	Data	See Fig. 2.12	
ROR	<i>dest,count</i>	D1 0E 00 10	ROR MEMWDS,1 ^c	Data	See Fig. 2.12	See Fig. 2.12
		D2 0E 04 10	ROR MEMBDS,CL ^d	Data	See Fig. 2.12	

8086 - Instruction Set – Shift and Rotate Instructions

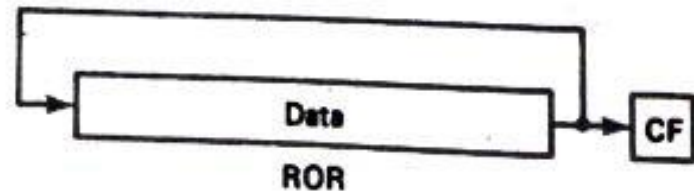
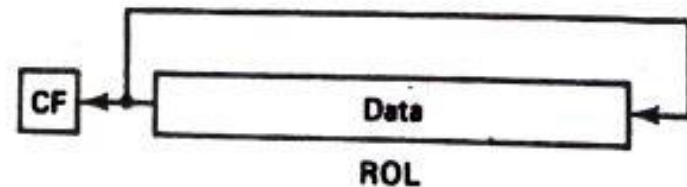
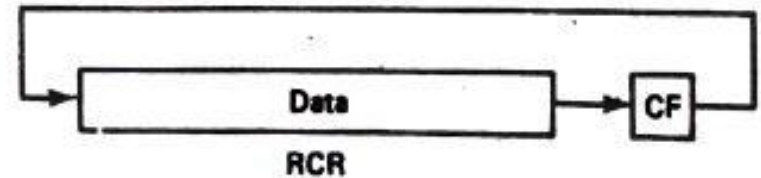
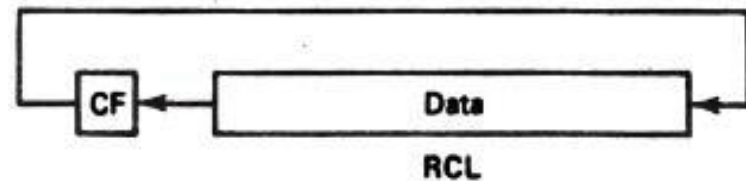
- Rotate Instructions

9. ROL

10. ROR

11. RCL

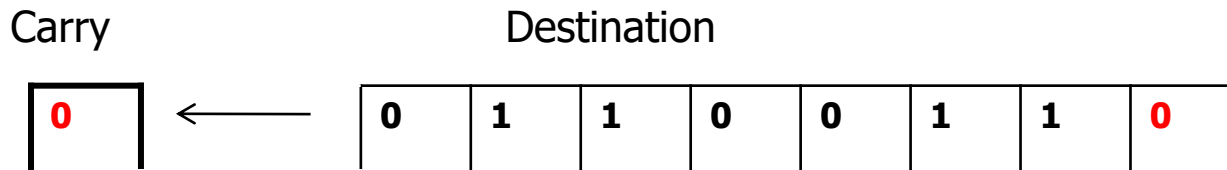
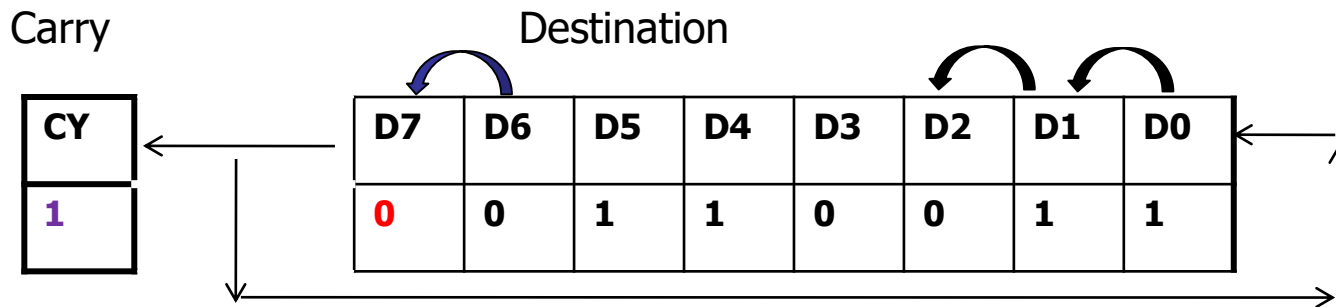
12. RCR



8086 - Instruction Set – Shift and Rotate Instructions

Eg: Count = 1, ROL BL,1

Assume before operation BL=0011 0011 & CF=1



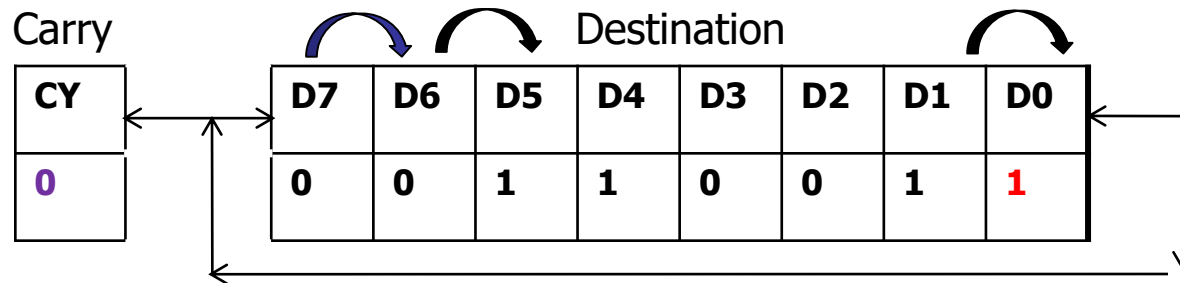
Eg: Count > 1

```
MOV CL, 05H
ROL BL,CL
```

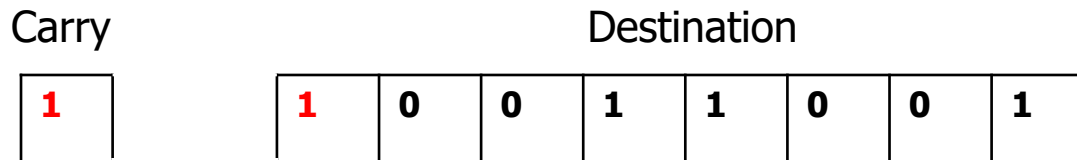
8086 - Instruction Set – Shift and Rotate Instructions

Eg: If Count = 1, ROR BL,1

Assume before operation BL=0011 0011 & CF=0



After operation BL=0001 1001 & CF=1



Eg: Count >1

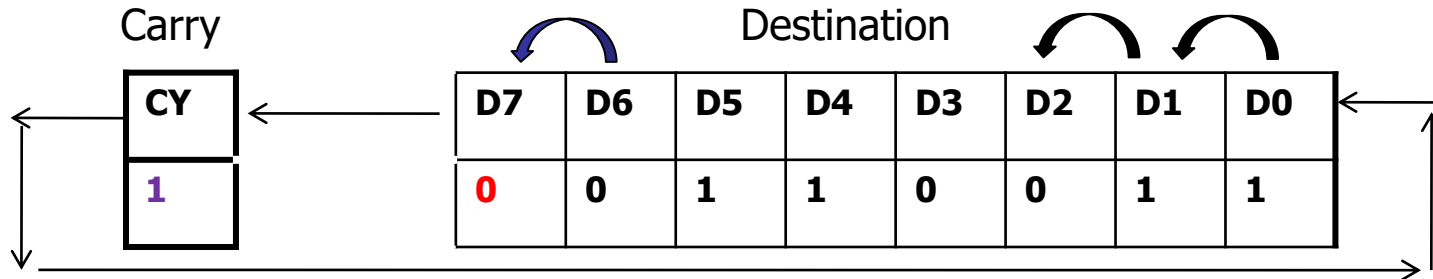
MOV CL, 05H

ROR BL,CL

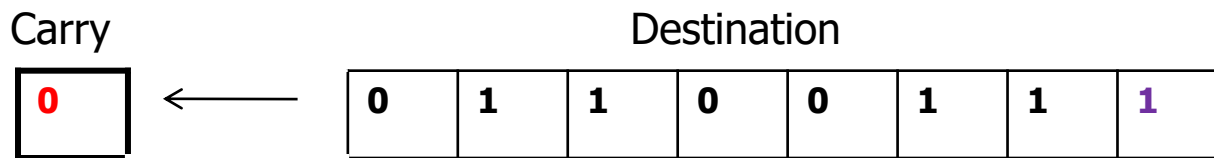
8086 - Instruction Set – Shift and Rotate Instructions

Eg: If Count = 1, RCL BL,1

Assume before operation BL=0011 0011 & CF=1



After operation BL=0110 0110 & CF=0



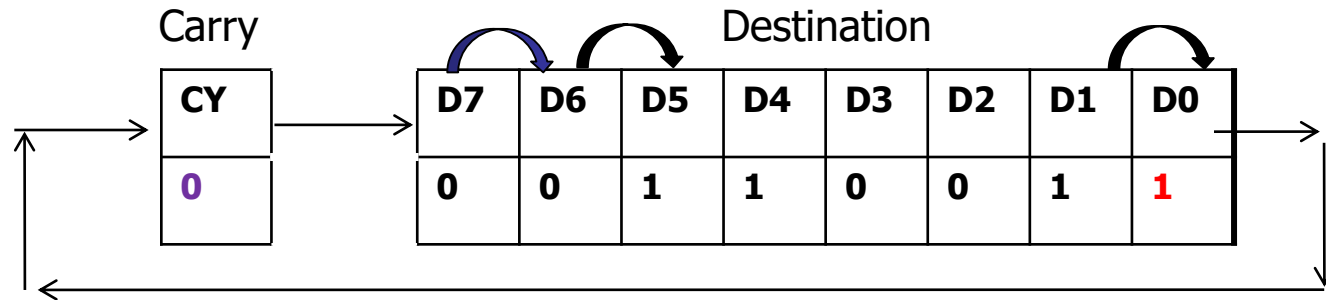
Eg: Count >1

MOV CL, 05H
RCL BL,CL

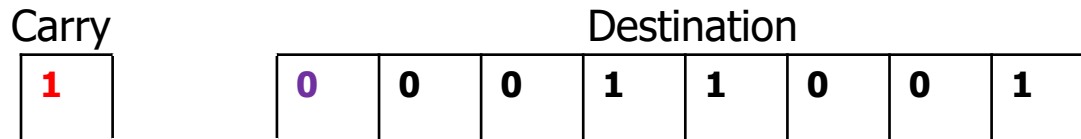
8086 - Instruction Set – Shift and Rotate Instructions

Eg: Count = 1 ;RCR BL,1

Assume before operation BL=0011 0011 & CF=0



After operation BL=0001 1001 & CF=1



Eg: Count >1

MOV CL, 05H

RCR BL,CL

