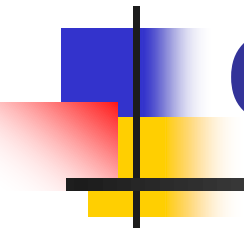


CSC405 Microprocessor

8086 Microprocessor

Instruction Set

rithesh.kini@thadomal.org

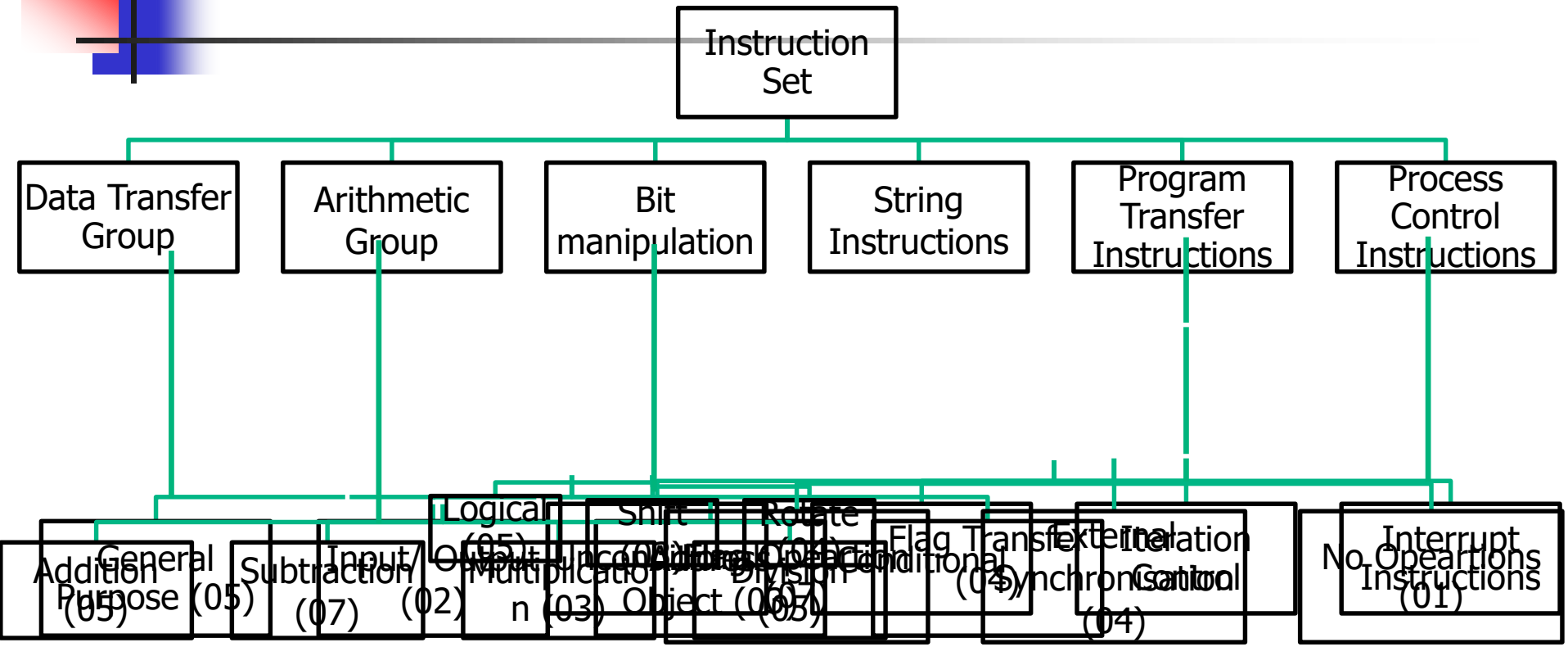
8086 - Instruction Set

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

rithesh.kini@thadomal.org

8086 - Instruction Set



rithesh.kini@thadomal.org

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.

rithesh.kini@thadomal.org

8086 - Instruction Set – Data Transfer Instructions

General Purpose Byte or Word Transfer Instruction

1. MOV : copy a Word or a Byte

MOV destination, source — destination ⟵ source

MOV operand 1, operand 2 — 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

rithesh.kini@thadomal.org

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**

rithesh.kini@thadomal.org

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**

rithesh.kini@thadomal.org

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.

rithesh.kini@thadomal.org

8086 - Instruction Set – Special Data Transfer Instructions

3. XLAT: Translate or Replace a byte

- XLAT/XLATB
- AL = DS:[BX+ 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

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

rithesh.kini@thadomal.org

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 offfset 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]

rithesh.kini@thadomal.org

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

rithesh.kini@thadomal.org

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.

rithesh.kini@thadomal.org

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.

rithesh.kini@thadomal.org

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

rithesh.kini@thadomal.org

8086 - Instruction Set – Special Data Transfer Instructions

5.2. LDS reg, Src: Load pointer using DS

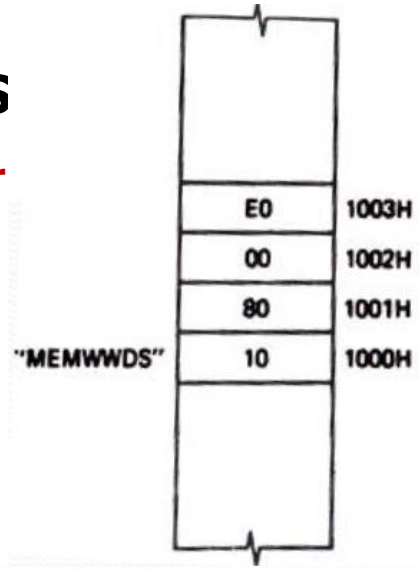
- The **source is always memory location** & **DS is used as segment reg** for memory.
- It is a **2 byte** instruction.
  i. It copies a word from 2 memory locations into reg specified in the instruction.
  ii. It copies a word from the next 2 memory locations into DS reg.
- It loads **32-bit pointer**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

rithesh.kini@thadomal.org

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]}

rithesh.kini@thadomal.org

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**.

rithesh.kini@thadomal.org

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]}

rithesh.kini@thadomal.org

8086 - Instruction Set – Special Data Transfer Instructions

5.3. LES reg, Src: Load pointer using ES

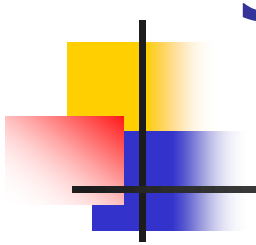
- The **offset addr**. is placed in the **destination register** and the **segment addr** is placed in **ES**.
- This instruction is very similar to LDS except that it initializes ES instead of DS.
- **E.g.**: LES BX, MEMWWDS

BX ⟵ {DS: [MEMWWDS], DS:[MEMWWDS +1]}

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

rithesh.kini@thadomal.org

8086 - Instruction Set – Special Data Transfer Instructions



6. Stack memory related instruction:

6.1 PUSH source

6.2 POP destination

rithesh.kini@thadomal.org

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.

rithesh.kini@thadomal.org

8086 - Instruction Set – Special Data Transfer Instructions

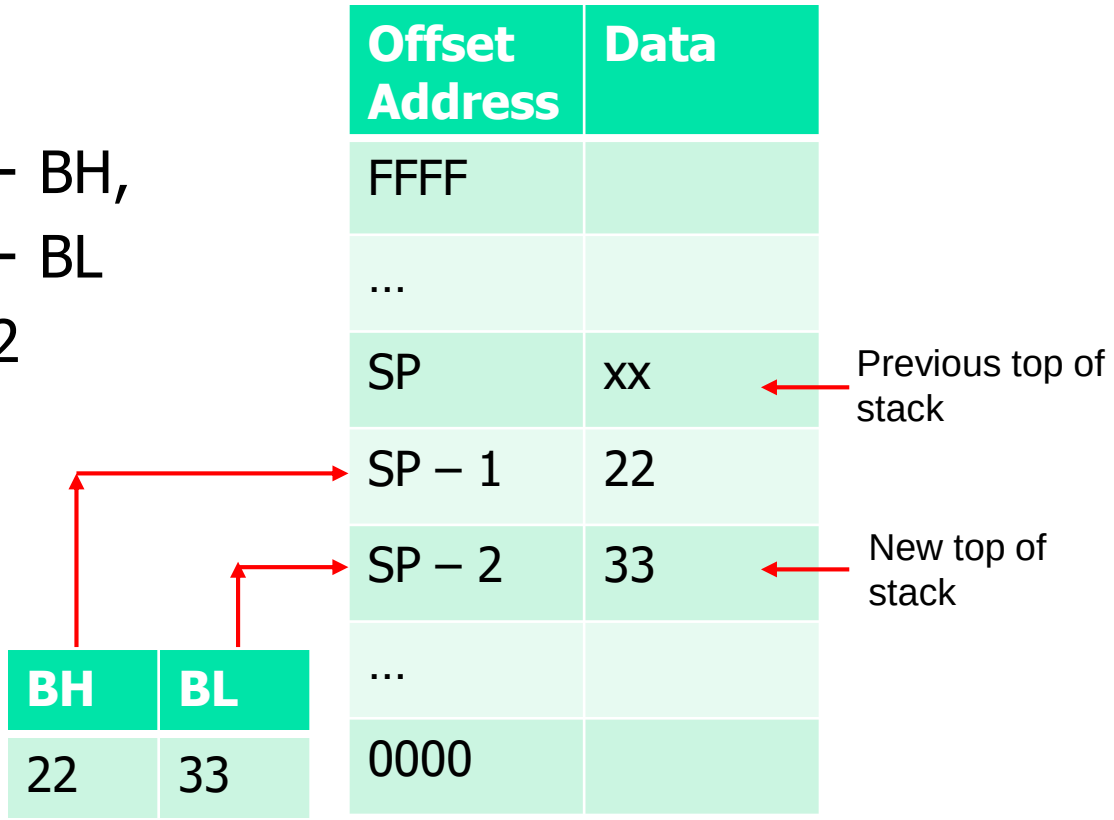
Example:

PUSH BX

SS:[SP-1] ← BH,

SS:[SP-2] ← BL

SP ← SP – 2



rithesh.kini@thadomal.org

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.

rithesh.kini@thadomal.org

8086 - Instruction Set – Special Data Transfer Instructions

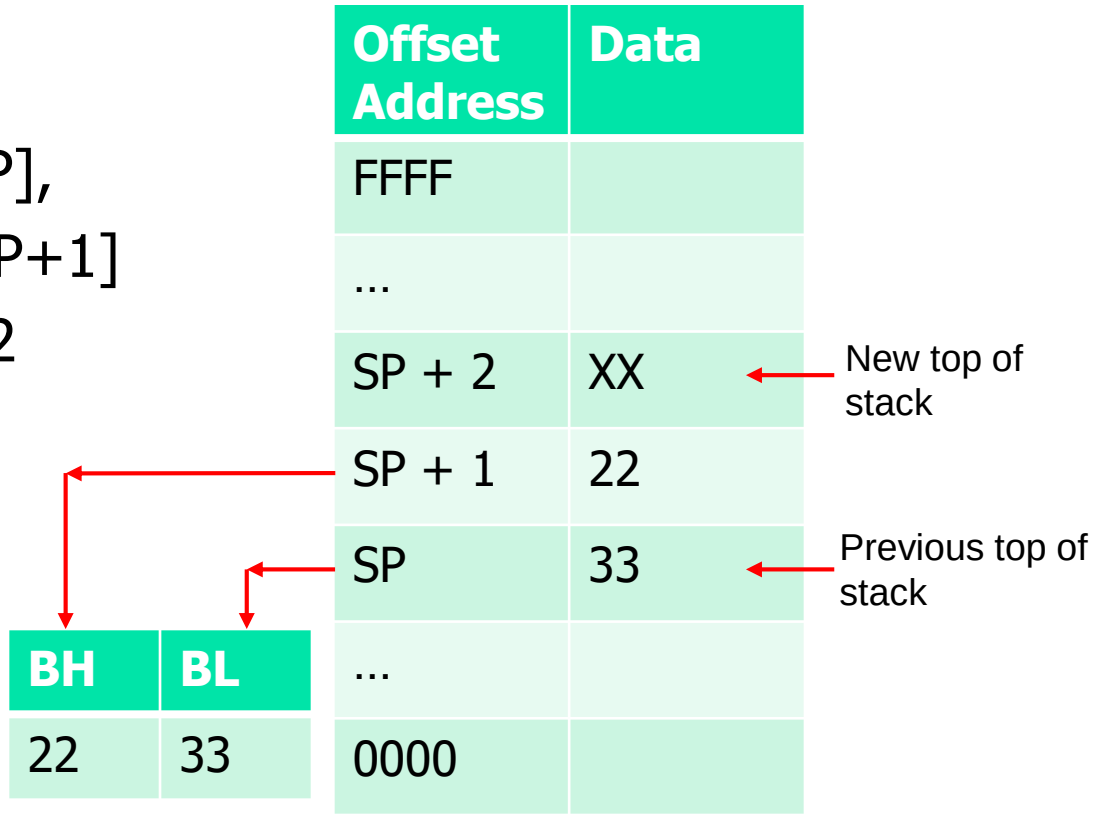
Example:

POP BX

BL ← SS:[SP],

BH ← SS:[SP+1]

SP ← SP + 2



rithesh.kini@thadomal.org

8086 - Instruction Set – Special Data Transfer Instructions

7. Flag Instructions

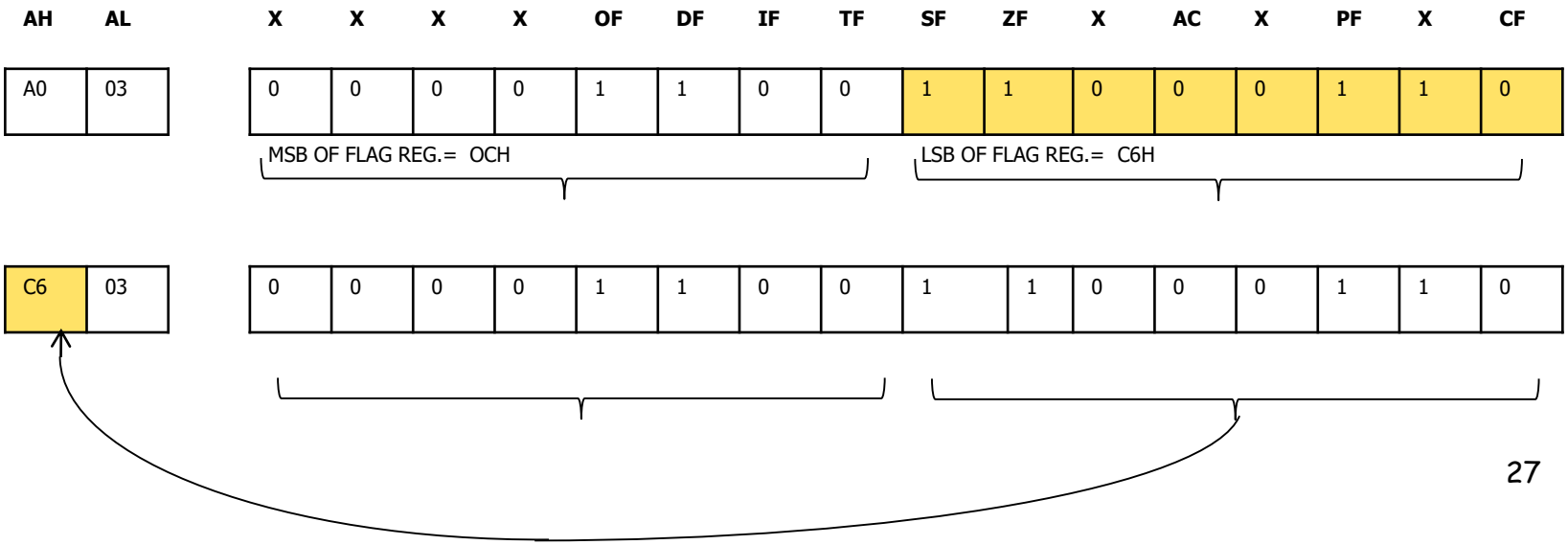
- These instructions are related to movement of flag register to/from a register & memory
- 4 instructions:
  - **LAHF**
  - **SAHF**
  - **PUSHF**
  - **POPF**

rithesh.kini@thadomal.org

8086 - Instruction Set – Special Data Transfer Instructions

7.1. LAHF

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

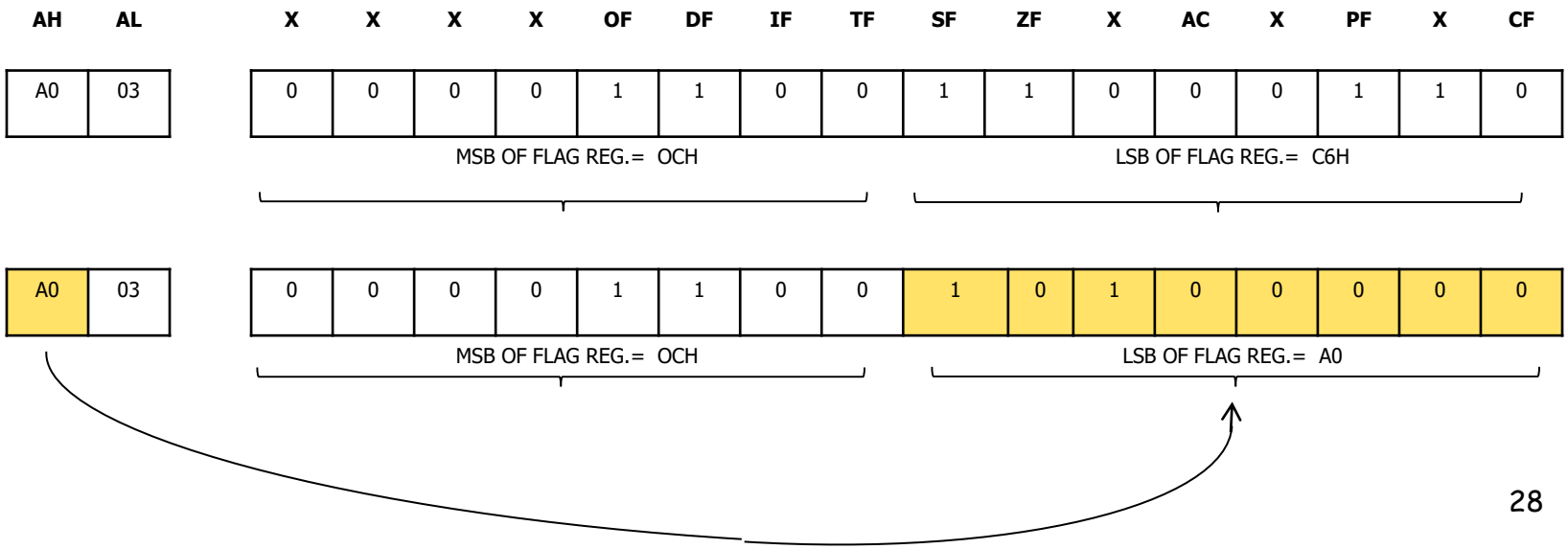


rithesh.kini@thadomal.org

8086 - Instruction Set – Special Data Transfer Instructions

7.2. SAHF

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



rithesh.kini@thadomal.org

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.

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

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

$SP \leftarrow SP - 2$

rithesh.kini@thadomal.org

8086 - Instruction Set – Special Data Transfer Instructions

7.4 POPF

Pop a word from the stack into the Flag register.

$FLAG_L \leftarrow SS:[SP]$,

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

$SP \leftarrow SP + 2$

rithesh.kini@thadomal.org

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

rithesh.kini@thadomal.org

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

rithesh.kini@thadomal.org

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

rithesh.kini@thadomal.org

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

rithesh.kini@thadomal.org

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

rithesh.kini@thadomal.org

8086 - Instruction Set – Arithmetic Instructions

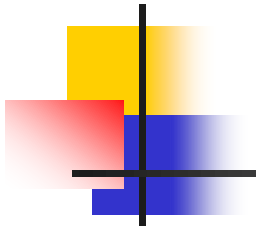
2) ADC Des, Src:

- 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

rithesh.kini@thadomal.org

8086 - Instruction Set – Arithmetic Instructions



32 bit Addition

```
    BX:AX
+   DX:CX
-----------
    DX:CX
```

ADD CX, AX

ADC DX, BX

rithesh.kini@thadomal.org

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

rithesh.kini@thadomal.org

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

rithesh.kini@thadomal.org

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

rithesh.kini@thadomal.org

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

rithesh.kini@thadomal.org

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
- **<u>Flags affected:</u>** AF, OF, PF, SF, ZF,

  CF = 1 EXCEPT WHEN THE OPERAND IS ZERO
- Assume AL= 0000 0101 = 05H then

  NEG AL; AL ⟵ 1111 1011 = FBH

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

rithesh.kini@thadomal.org

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

rithesh.kini@thadomal.org

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; AX = AL * BL

  MUL BX; DX : AX = AX * BX

rithesh.kini@thadomal.org

8086 - Instruction Set – Arithmetic Instructions

General mnemonic		Object code	Mnemonic	Segment for memory access	Symbolic operation	Description
Op-code	Operand[a]					
MUL	*source*	F6 E3	MUL BL	Within CPU	AX ← 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 ← AX*CX	
			MUL BYTE PTR[BX]	Data	AX ← AL*[BX]	
		F6 27	MUL MEMWDS[b]	Data	DX:AX ← AX*[1001H:1000H]	
		F7 26 00 10				

rithesh.kini@thadomal.org

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

rithesh.kini@thadomal.org

8086 - Instruction Set – Arithmetic Instructions

General mnemonic Op-code	Operand[a]	Object code	Mnemonic	Segment for memory access	Symbolic operation	Description
DIV	*source*	F6 F3	DIV BL	Within CPU	AX ←AX/BL	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
		F7 F1	DIV CX	Within CPU	DX:AX ← AX/CX	
		F6 37	DIV BYTE PTR[BX]	Data	AX ←AX/[BX]	
		F7 36 00 10	DIV MEMWDS[b]	Data	DX:AX ← AX/[1001H:1000H]	

Ex: DIVIDING 65,000 BY 2

rithesh.kini@thadomal.org

8086 - Instruction Set – Arithmetic Instructions

General mnemonic		Object code	Mnemonic	Segment for memory access	Symbolic operation	Description
Op-code	Operand[a]					
IMUL	*source*	F6 EB	IMUL BL	Within CPU	AX ← 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 ← AX*CX (signed)	
		F6 2F	IMUL BYTE PTR[BX]	Data	AX ← AL*[BX] (signed)	
		F7 2E 00 10	IMUL MEMWDS[b]	Data	DX:AX ← AX*[1001H:1000H] (signed)	

rithesh.kini@thadomal.org

8086 - Instruction Set – Arithmetic Instructions

General mnemonic		Object code	Mnemonic	Seqment for memory access	Symbolic operation	Description
Op-code	Operand[a]					
IDIV	*source*	F6 FB	IDIV BL	Within CPU	AX ← AL/BL (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*
		F7 F9	IDIV CX	Within CPU	DX:AX ← AX/CX (signed)	
		F6 3F	IDIV BYTE PTR[BX]	Data	AX ← AL/[BX] (signed)	
		F7 3E 00 10	IDIV MEMWDS[b]	Data	DX:AX ← AX/[1001H:1000H] (signed)	

rithesh.kini@thadomal.org

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?

rithesh.kini@thadomal.org

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)

rithesh.kini@thadomal.org

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

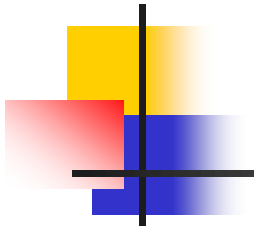
    then AL =AL + 06, AF=1

- 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

rithesh.kini@thadomal.org

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

rithesh.kini@thadomal.org

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

rithesh.kini@thadomal.org

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

rithesh.kini@thadomal.org

8086 - Instruction Set – Arithmetic Adjust Instruction

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

- Executed after adition of 2 ASCII data to convert the result in AL to correct unpacked BCD form.
- AAA instrcution woks 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 ←1; CF ←1

  AL <- AL.0F

Ex:

MOV AH,0;	AH = 0000 0000
MOV AL,7;	AL = 0000 0111
MOV BL, 5;	BL = 0000 0101
ADD AL, BL;	AL = 0000 1100 (0CH)
AAA;	AX = 0102H
ADD AX, 3030H;	AX = 3132H

rithesh.kini@thadomal.org

8086 - Instruction Set – Arithmetic Adjust Instruction

16. AAS (ASCII Adjust after Subtraction):

- Similar like AAA
- It 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

rithesh.kini@thadomal.org

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 = 0036H

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.

rithesh.kini@thadomal.org

General mnemonic		Object code	Coding example			
Op-code	Operand		Mnemonic	Segment for memory access	Symbolic operation	Description
AAM	none	D4 0A	AAM	Within CPU	AH ← AL/0AH AL ← 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
AAD	none	D5 0A	AAD	Within CPU	AL ← (AH * 0AH) + AL AH ← 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

rithesh.kini@thadomal.org

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

rithesh.kini@thadomal.org

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

rithesh.kini@thadomal.org

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 **1**001 1000

  Then CBW gives AX= **1111 1111 1**001 0001

rithesh.kini@thadomal.org

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 = **1**000 0000 1001 0001

  DX = xxxx xxxx xxxx xxxx
- Then CWD gives AX = **1**000 0000 1001 0001

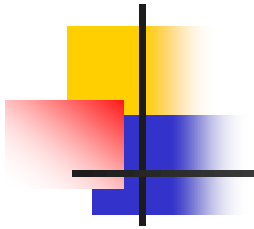
  DX = **1111 1111 1111 1111**

rithesh.kini@thadomal.org

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

rithesh.kini@thadomal.org



rithesh.kini@thadomal.org