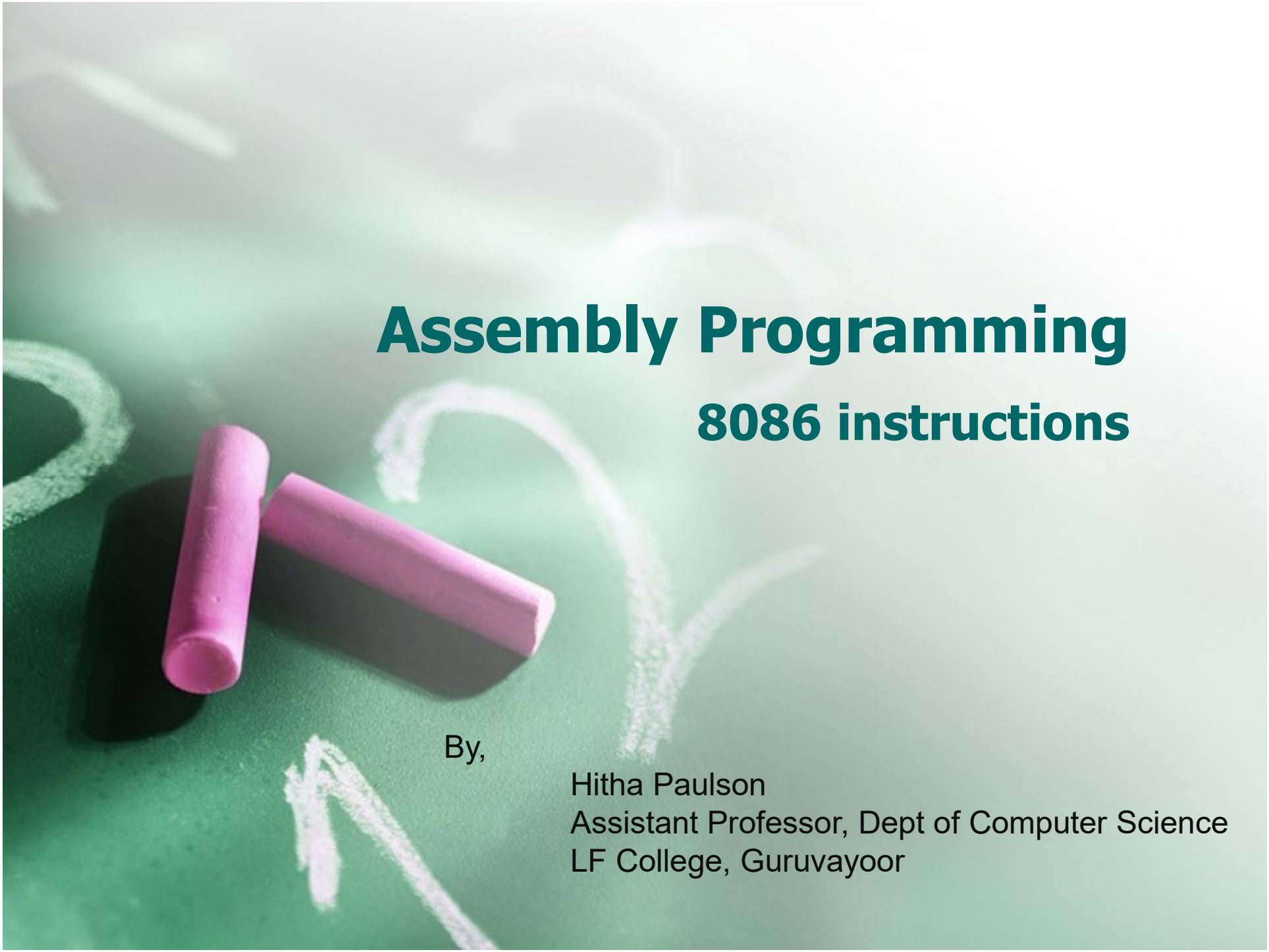
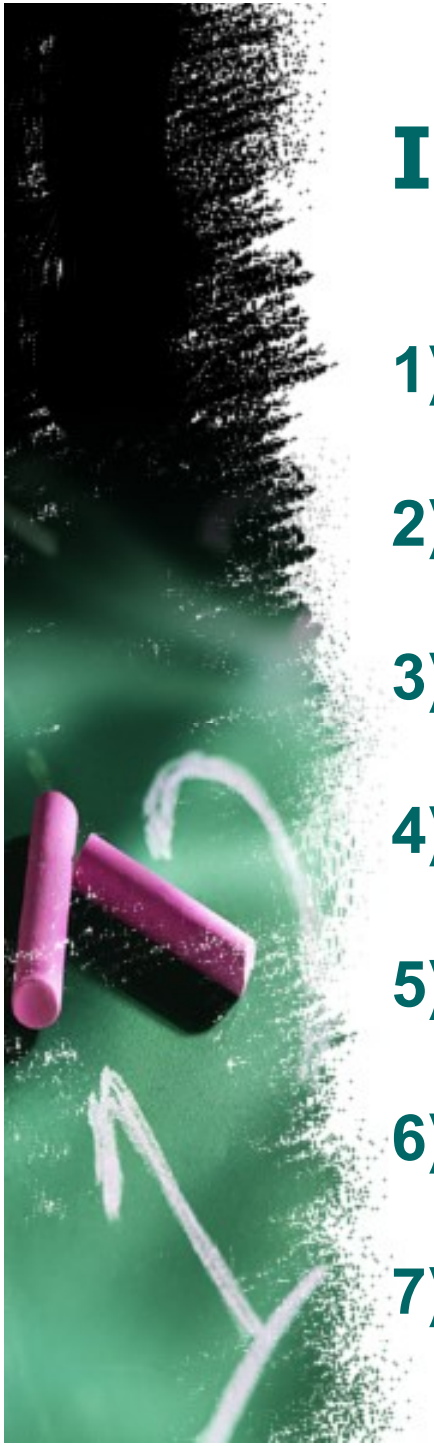


Assembly Programming

8086 instructions

By,

Hitha Paulson
Assistant Professor, Dept of Computer Science
LF College, Guruvayoor



Instruction – Functional groups

1) Data transfer instructions
2) Arithmetic and logical instructions
3) Shift and rotate instructions
4) Branch & loop instructions
5) Flag Manipulation Instructions
6) Machine control instructions
7) String operation instructions

Following are the conventions used in describing the instructions.

Convention	Meaning
src	source
dest	destination
reg	register
r/m	register or a memory location
r/m16	16 bit register or a word memory location
immed16	immediate 16 bit number
addr	address of a memory location
reg8	8-bit register
reg16	16-bit register
src8	source of type 8-bit
src16	source of type 16-bit
mem16	16-bit memory location
MOD	modulus or a remainder
~	logical NOT operation
&	logical AND operation
|	logical OR operation
^	logical EX-OR (exclusive OR) operation
←	assignment operation
≠	not equal

Data transfer instructions

- MOV dest, src
- Copy operand2 to operand1.

The MOV instruction <u>cannot</u>:

- set the value of the CS and IP registers.
- copy value of one segment register to another segment register (should copy to general register first).
- copy immediate value to segment register (should copy to general register first).
- Algorithm:
  operand1 = operand2

Data transfer instructions

XCHG dest,src

- Exchange values of two operands.
- Both cant be memory.
- Immediate data not allowed
- Algorithm:

  operand1 < - > operand2

Data transfer instructions

- IN accumulator , port
- Reads data from port

- Direct or indirect
- Indirect use DX for port number

Data transfer instructions

- OUT port, accumulator
- Outs accumulator data to a port

- Direct or indirect
- Indirect use DX for port number

Data transfer instructions

- XLAT/XLATB – translate a byte
- Translate byte from table.
  Copy value of memory byte at DS:[BX + unsigned AL] to AL register.

  Algorithm:

  AL = DS:[BX + unsigned AL]

Data transfer instructions

- LEA – Load Effective Address

- Load Effective Address.

  Algorithm:

  REG = address of memory (offset)

- LEA BX, ADR = MOV BX,OFFSET ADR

Data transfer instructions

- LDS – Load register and DS register
- LES – Load register and ES register
- Load memory double word into word register and DS.

  Algorithm: REG = first word DS = second word
- Example:
- LDS BX,5000H

Data transfer instructions

- LAHF – Load AH with Flags
- Load AH from 8 low bits of Flags register.

  Algorithm:

  AH = flags register
  AH bit: 7 6 5 4 3 2 1 0 [SF] [ZF] [0] [AF] [0] [PF] [1] [CF]
- SAHF – Store AH in to lower byte of Flag register

Data transfer instructions

- PUSH src
- PUSH immed
- PUSHA
- PUSHF
- POP dest
- POPA
- POPF

Arithmetic and logical instructions

- ADD – Addition

- ADD dest, src
- Memory memory addition not possible
- Segment registers cant be added
- Flags are affected

- Algorithm:

  operand1 = operand1 + operand2

Arithmetic and logical instructions

- **ADC – Add with carry**

- ADC dest, src
- Flags affected

- Algorithm:

  operand1 = operand1 + operand2 + CF

Arithmetic and logical instructions

- **INC – increment**

- INC dest
- Immediate data is not an operand

- Algorithm:

  operand = operand + 1

- *Carry flag will not be affected*

Arithmetic and logical instructions

- **DEC – Decrement**
- DEC dest
- Algorithm:

  operand = operand – 1
- *Carry flag will not be affected*
-

Arithmetic and logical instructions

- **SUB – Subtraction**

- SUB dest, src
- Algorithm:

  operand1 = operand1 - operand2

- Both operands cant be memory

- Destination cant be immediate

Arithmetic and logical instructions

- **SBB – Subtract with borrow**

- SBB dest, src
- Algorithm:

  operand1 = operand1 - operand2 - CF

Arithmetic and logical instructions

- **CMP – Compare**
- CMP dest, src
- Algorithm:

  operand1 - operand2
  (dest – src)
- Dest cant be immediate
-

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

CMP function

- 1. if(dest > src)
  subtraction requires no borrow
  CF = 0, ZP= 0, SF= 0.

- 2. if(dest < src)
  subtraction requires borrow
  CF= 1, ZF= 0, SF= 1.

- 3. if(dest= arc)
  result of subtraction is zero hence ZP flag is set
  CF= 0, ZF = 1, SF= 0.

Arithmetic and logical instructions

- **NEG –Makes operand negative (two's complement).**

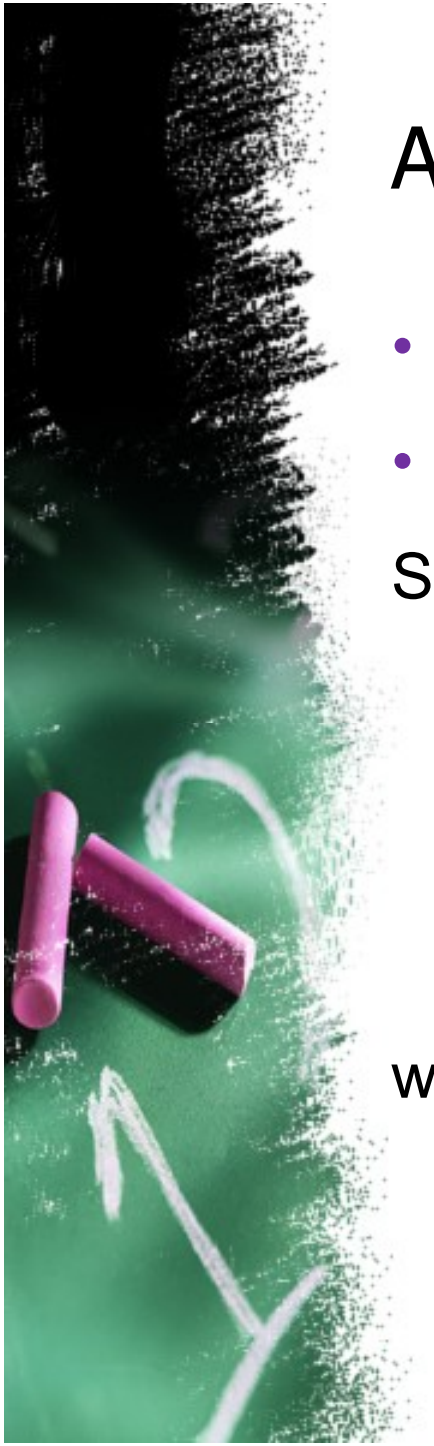
- NEG dest
- Algorithm: Invert all bits of the operand
- Add 1 to inverted operand
- **EG:-** MOV AL, 5 ; AL = 05h
- NEG AL ; AL = 0FBh (-5)
- NEG AL ; AL = 05h (5)

Arithmetic and logical instructions

- **MUL – Multiplication**
- **MUL src**
- Unsigned multiply.
- Operand cant be immediate

- Algorithm:
- when operand is a **byte**: AX = AL * operand.
- when operand is a **word**:

(DX AX) = AX * operand.



Arithmetic and logical instructions

- **IMUL – Multiplication**
- **IMUL SRC**

Signed multiply.

Algorithm:

when operand is a **byte**:

AX = AL * operand.

when operand is a **word**:

(DX AX) = AX * operand.

Arithmetic and logical instructions

- **DIV – Division**
- **DIV src**

Unsigned divide.

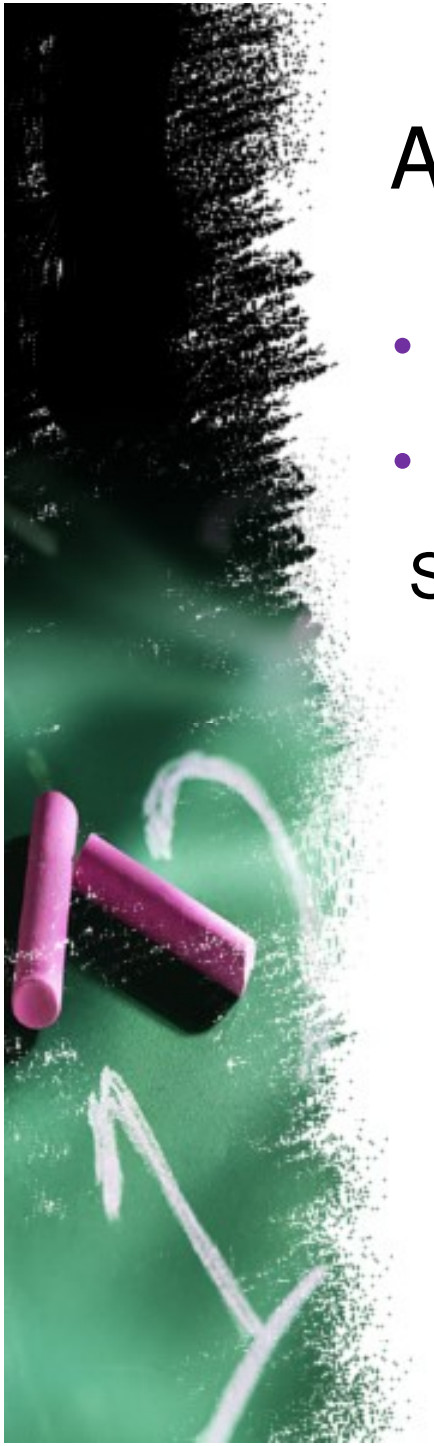
Algorithm:

When operand is a **byte**:

AL = AX / operand
AH = remainder (modulus)

when operand is a **word**:

AX = (DX AX) / operand
DX = remainder (modulus)



Arithmetic and logical instructions

- **IDIV – Integer Division**
- **IDIV src**

Signed divide.

Algorithm:

when operand is a **byte**:

AL = AX / operand
AH = remainder (modulus)

when operand is a **word**:

AX = (DX AX) / operand
DX = remainder (modulus)

Arithmetic and logical instructions

- **CBW – Convert signed Byte to Word**
- **Sign extended number**
- Convert byte into word.

  Algorithm:

  if high bit of AL = 1 then: AH = FFh
-

  else AH = 0

Arithmetic and logical instructions

- **CWD – Convert Word to double word**
- **Sign extended number**
- Algorithm:

  if high bit of AX = 1 then: DX =FFFFh
-

  else DX = 0

Arithmetic and logical instructions

- DAA - Decimal adjust After Addition.
  Decimal adjust Accumulator.
- Corrects the result of addition of two **packed BCD values.**

  Algorithm:

  if low nibble of AL > 9 or AF = 1 then:
  AL = AL + 6
- After this if upper nibble of AL is greater than 9 or if carry flag is set add 60h to AL.

EXAMPLE

- **ADD AL,CL** AL=53 , CL =29

  DAA AL=73, CL = 29

Arithmetic and logical instructions

- **DAS - Decimal adjust After Subtraction.**
  Corrects the result of subtraction of two packed BCD values.

  Algorithm:

  if low nibble of AL > 9 or AF = 1 then:
  AL = AL - 6
- After this if upper nibble of AL is greater than 9 or if carry flag is set subtract 60h from AL.

EXAMPLE

- **SUB AL,CL** AL=75 , CL =46

  DAA

Arithmetic and logical instructions

- **AAA - ASCII Adjust after Addition.**
  Corrects result in AH and AL after addition when working with un packed BCD values.

  Algorithm:

  1. clear the higher order nibble of AL
     - *(AL=AL & 0FH)*

  2. if low nibble of AL > 9 or AF = 1 then:
     a. AL = AL + 6 b. AH = AH + 1
     c. AF = 1 , CF = 1
  3. *Clear high order nibble of AL (AL = AL& 0FH)*

Arithmetic and logical instructions

- **AAS -** ASCII Adjust after Subtraction.
  Corrects result in AH and AL after subtraction when working with unpacked BCD values.

  Algorithm:

  1. clear the higher order nibble of AL
     - *(AL=AL & 0FH)*

  2.if low nibble of AL > 9 or AF = 1 then:

  a. AL = AL - 6 b. AH = AH - 1

  c. AF = 1 , CF = 1

  3. Clear high order nibble of AL (AL = AL& 0FH)

Arithmetic and logical instructions

- **AAM -** ASCII Adjust after Multiplication. Corrects the result of multiplication of two BCD values.

  Algorithm:

  AH = AL / A h
- AL = remainder
-

Arithmetic and logical instructions

- **AAD -** ASCII Adjust before Division.
  Prepares two BCD values for division.
- BCD to binary conversion
- Algorithm:

  AL = (AH * 10) + AL
- AH = 0
-

Arithmetic and logical instructions

- **AND – Logical AND**

- AND dest , src
- Dest = dest & src
- Mask
- Reset dest bits which are zeros in src

Arithmetic and logical instructions

- **TEST – test and set flags**

- TEST dest , src
- Non destructive AND

Arithmetic and logical instructions

- **OR – Logical OR**

- OR dest , src
- Set dest bits which are in src

Arithmetic and logical instructions

- **XOR – Logical Exclusive OR**

- XOR dest , src
- Toggle dest bits which are in src

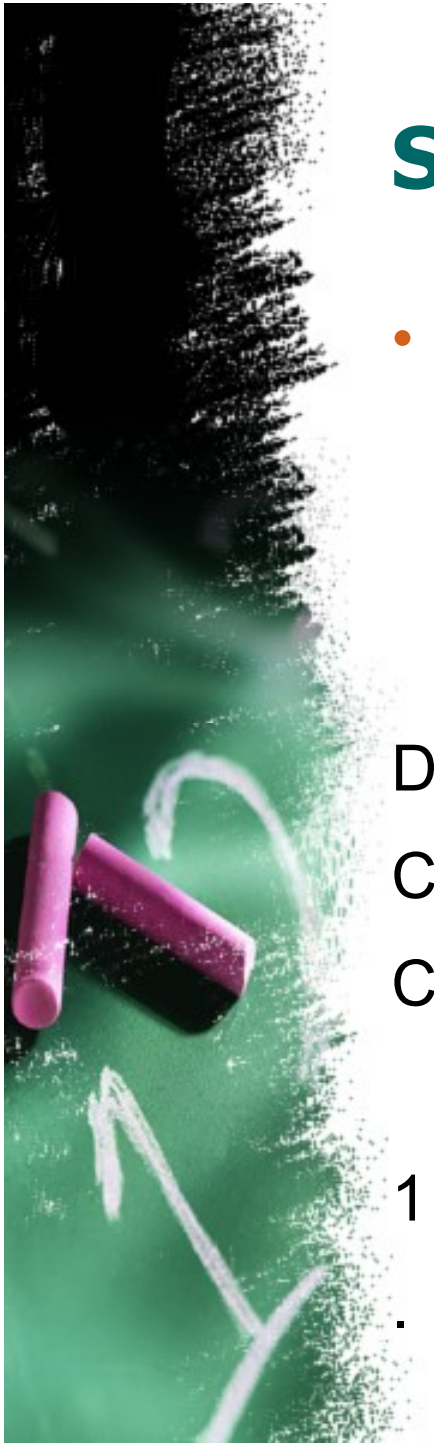
Arithmetic and logical instructions

- **NOT – Logical NOT**

- NOT dest
- Toggle dest bits (converts dest to its 1’s complement)

Shift and rotate instructions

- SHL: Shift Logical Left
- SHR: Shift Logical Right
- SAL: Shift Arithmetic Left
- SAR: Shift Arithmetic Right
- ROL: Rotate Left
- ROR: Rotate right
- RCL: Rotate through carry left
- RCR: Rotate through carry right



Shift and rotate instructions

- **SHL: Shift Logical Left**

  SHL dest, count

Dest = reg / mem

Count = reg / immed (if one)

Count range from 0 – 15

1 shift equivalent to multiply by two.

.

Shift and rotate instructions

- **SHR: Shift Logical Right**

SHR dest, count

Dest = reg / mem

Count = reg / immed (if one)

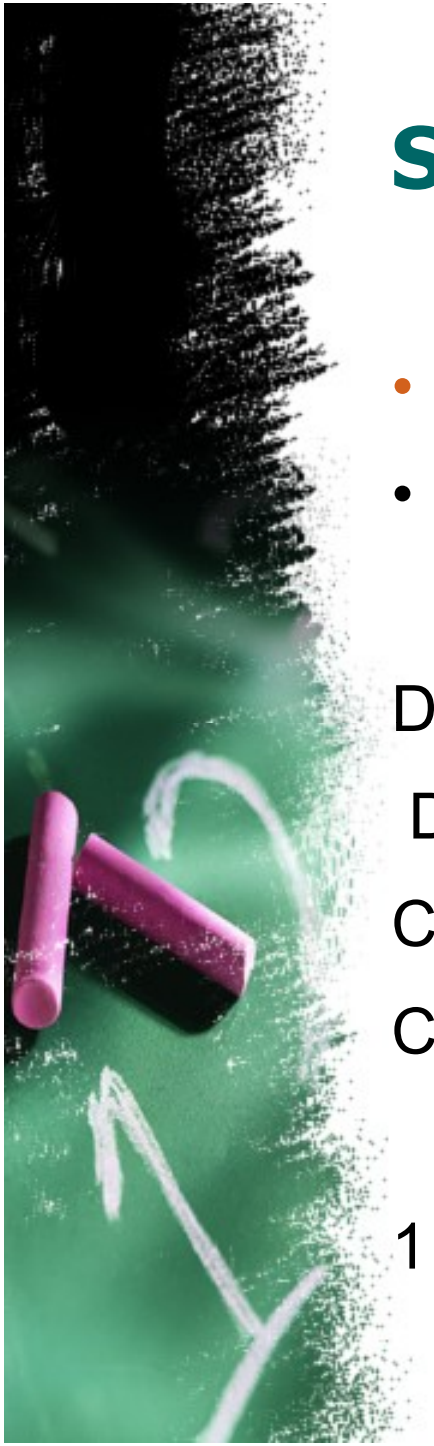
Count range from 0 – 15

1 shift equivalent to divide by two. (faster than DIV instruction)

.

Shift and rotate instructions

- **SAL: Shift Arithmetic Left**
- SAL dest, count
- Same as SHL



Shift and rotate instructions

- **SAR: Shift Arithmetic Right**
- SAR dest, count

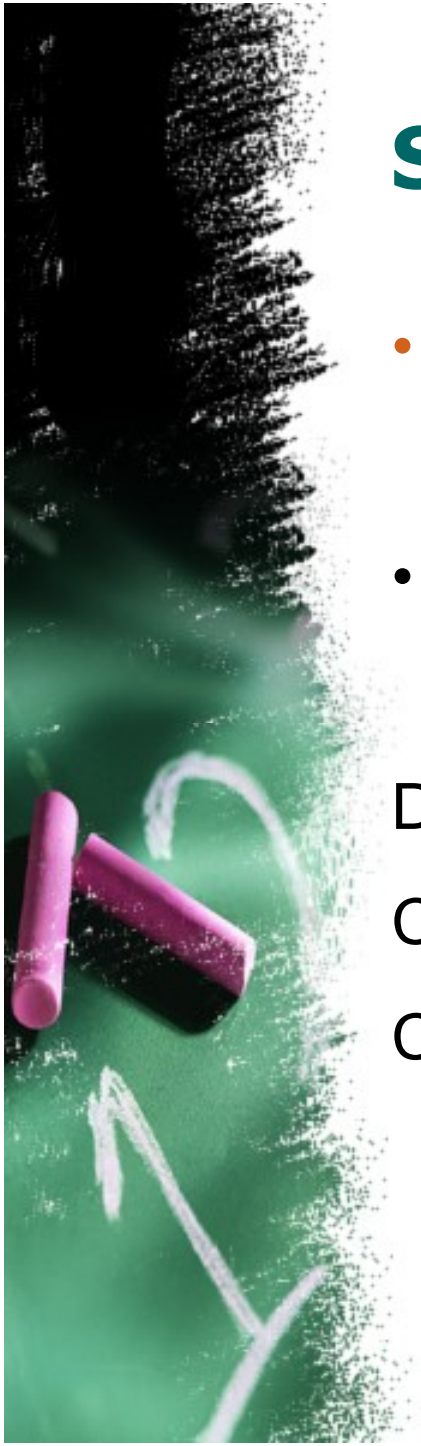
Dest should be signed

Dest = reg / mem

Count = reg / immed (if one)

Count range from 0 – 15

1 shift equivalent to divide by two. (faster than IDIV instruction)



Shift and rotate instructions

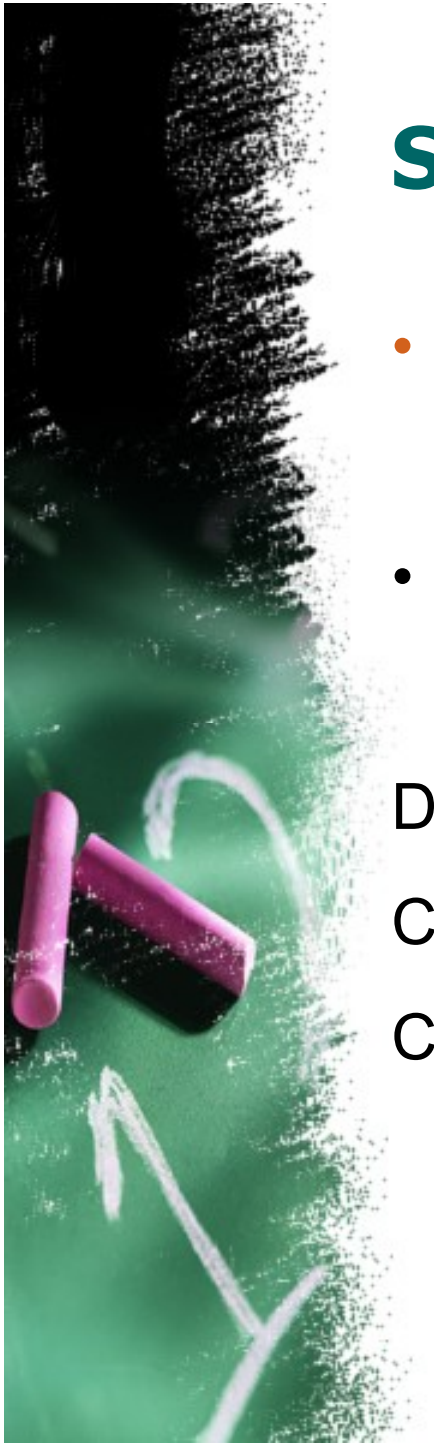
- **ROL: Rotate Left**

- ROL dest, count

Dest = reg / mem

Count = reg / immed (if one)

Count range from 0 – 15



Shift and rotate instructions

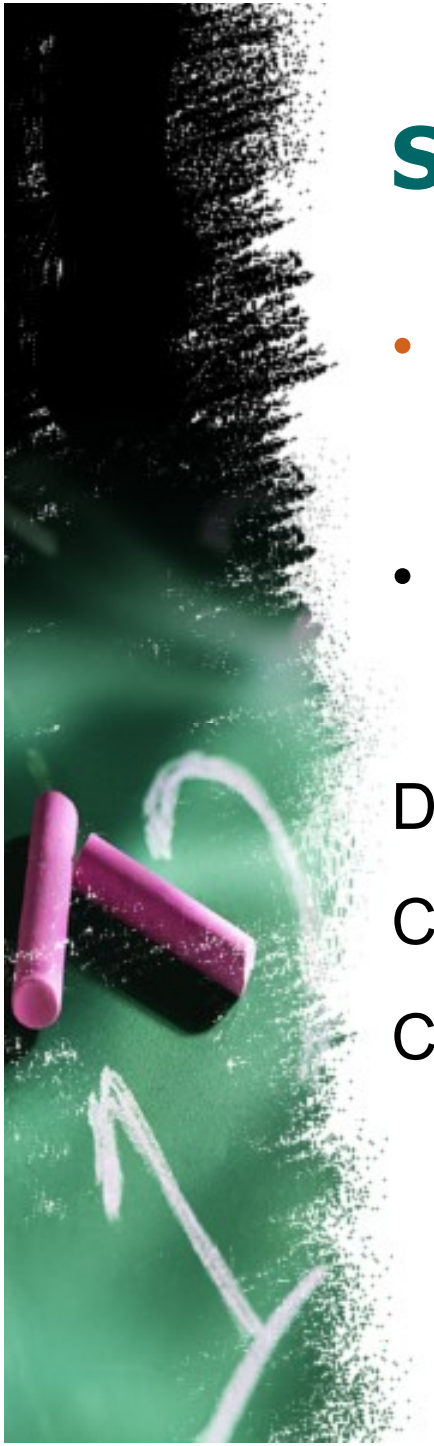
- **ROR: Rotate right**

- ROR dest, count

Dest = reg / mem

Count = reg / immed (if one)

Count range from 0 – 15



Shift and rotate instructions

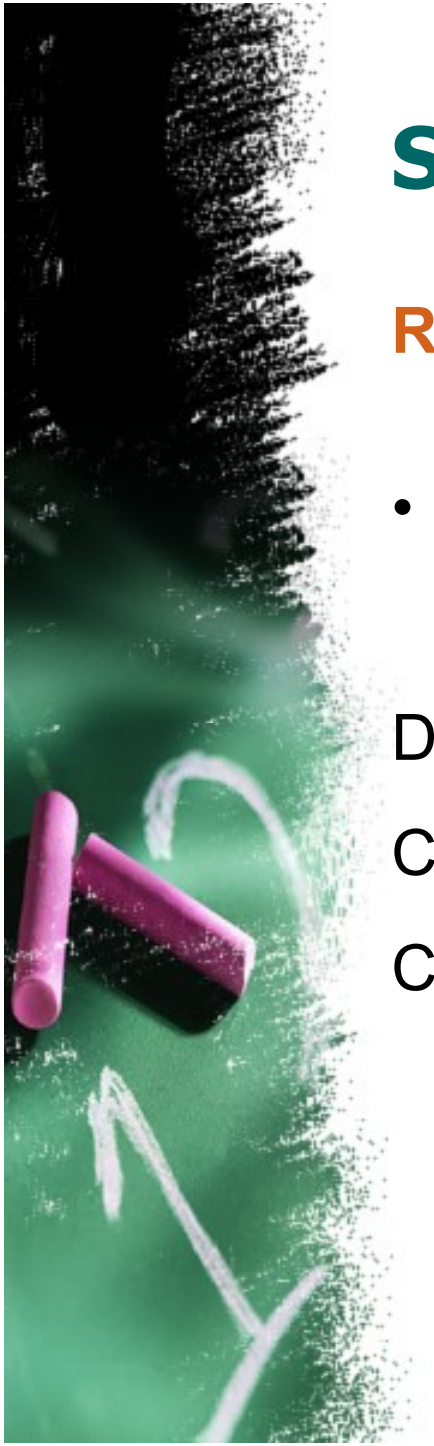
- **RCL: Rotate through carry left**

- RCL dest, count

Dest = reg / mem

Count = reg / immed (if one)

Count range from 0 – 15



Shift and rotate instructions

RCR: Rotate through carry right

- RCR dest, count

Dest = reg / mem

Count = reg / immed (if one)

Count range from 0 – 15

Branch instructions - Unconditional

- **CALL : Unconditional Call**
  - *Direct near*

    OPCODE DISP_LB DISP_HB
  - *Indirect near*

    OPCODE OPCODE indirect R/M
  - *Direct far*

    OPCODE OFFSET(LB,HB) SEGMENT(LB,HB)
  - *Indirect far*

    OPCODE OPCODE indirect R/M

Branch instructions - Unconditional

- **RET:** return from the procedure
  - *Pop operation from stack*
- **INT N** : Interrupt type N
  - *N*4 is the offset*
  - *0000h the segment address*
- **INTO:** interrupt on overflow (type 4)
- **IRET** (Return from Interrupt Service Routine)

Branch instructions - Unconditional

- JMP : jump
- JMP target

- JUMP 8-bit displacement
  - *Intrasegment,relative,near jump*
- JUMP 16-bit displacement
  - *Intrasegment,relative,Far jump*
- JUMP IP,CS
  - *Intersegment,directjump*

Branch instructions - Unconditional

- LOOP : Decrease CX, jump to label if CX not zero.
- LOOP short_lable
-

  Algorithm:

  CX = CX - 1

- if CX <> 0 then
  - *jump*
- else
  - *no jump, continue*

Branch instructions - conditional

- Only short jumps can be implemented (-128 to 127)

Table 2.3 *Conditional Branch Instructions*

	Mnemonic	*Displacement*	*Operation*
1.	JZ/JE	Label	Transfer execution control to address 'Label', if ZF=1.
2.	JNZ/JNE	Label	Transfer execution control to address 'Label', if ZF=0.
3.	JS	Label	Transfer execution control to address 'Label' , if SF=1.
4.	JNS	Label	Transfer execution control to address 'Label', if SF=0.
5.	JO	Label	Transfer execution control to address 'Label', if OF=1.
6.	JNO	Label	Transfer execution control to address 'Label', if OF=0.
7.	JP/JPE	Label	Transfer execution control to address 'Label', if PF=1.
8.	JNP	Label	Transfer execution control to address 'Label', if PF=0.
9.	JB/JNAE/JC	Label	Transfer execution control to address 'Label', if CF=1.
10.	JNB/JAE/JNC	Label	Transfer execution control to address 'Label', if CF=0.
11.	JBE/JNA	Label	Transfer execution control to address 'Label', if CF=1 or ZF=1.
12.	JNBE/JA	Label	Transfer execution control to address 'Label', if CF=0 or ZF=0.
13.	JL/JNGE	Label	Transfer execution control to address 'Label', if neither SF=1 nor OF=1.
14.	JNL/JGE	Label	Transfer execution control to address 'Label', if neither SF=0 nor OF=0.
15.	JLE/JNC	Label	Transfer execution control to address 'Label', if ZF=1 or neither SF nor OF is 1.
16.	JNLE/JE	Label	Transfer execution control to address 'Label', if ZF=0 or at least any one of SF and OF is 1(Both SF and OF are not 0).

Branch instructions - conditional

- JCXZ: Jump if CX register is ZERO
- JCXZ short_lable
- Algorithm:

  if CX = 0 then jump

Table 2.4 *Conditional Loop Instructions*

Mnemonic	*Displacement*	*Operation*
LOOPZ/LOOPE (Loop while ZF = 1; equal)	Label	Loop through a sequence of instructions from 'Label' while ZF=1 and CX ≠ 0.
LOOPNZ/LOOPENE (Loop while ZF = 0; not equal)	Label	Loop through a sequence of instructions from 'Label' while ZF=0 and CX ≠ 0.

Table 2.5 *Flag Manipulation Instructions*

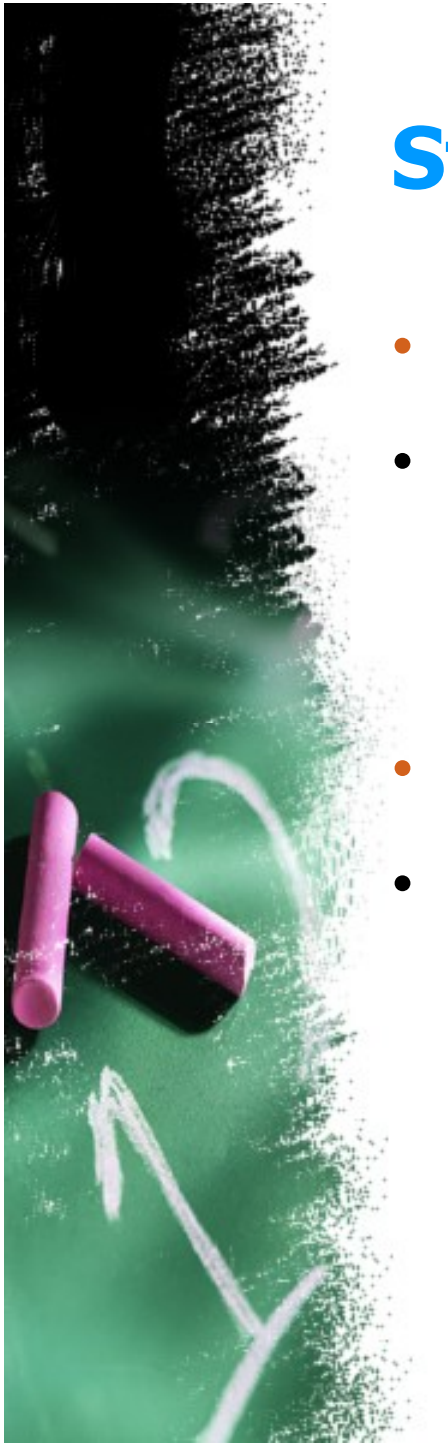
CLC	-	Clear carry flag
CMC	-	Complement carry flag
STC	-	Set carry flag
CLD	-	Clear direction flag
STD	-	Set direction flag
CLI	-	Clear interrupt flag
STI	-	Set interrupt flag

Table 2.6 *Machine Control Instructions*

WAIT	-	Wait for Test input pin to go low
HLT	-	Halt the processor
NOP	-	No operation
ESC	-	Escape to external device like NDP (numeric co-processor)
LOCK	-	Bus lock instruction prefix.

String operation instructions

- Series of data bytes → byte strings or word strings
- For referring a string 2 parameters are needed → starting or end address and length of a string.
- Length of the string stored in CX register.
- Increment or decrement depends on Direction flag.



String operation instructions

- **CLD – Clear direction flag**
- SI and DI will be incremented by chain instructions

- **STD – Set direction flag**
- SI and DI will be decremented by chain instructions:

String operation instructions

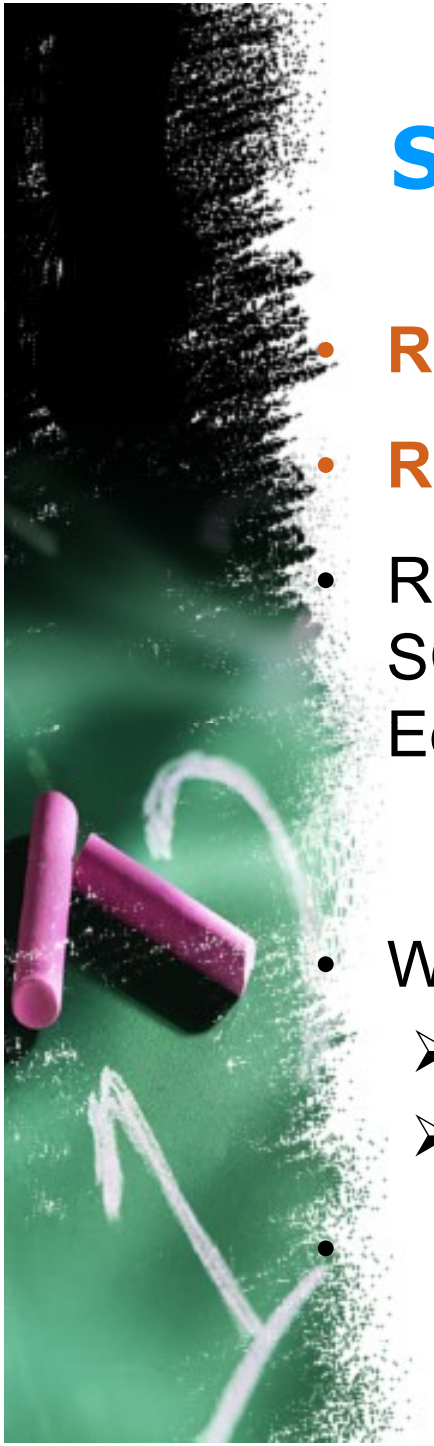
- **REP – Repeat**
- Repeat following MOVSB, MOVSW, LODSB, LODSW, STOSB, STOSW instructions CX times.

  Algorithm:

  while (CX<>0)
- Execute the string instruction
- CX = CX – 1

String operation instructions

- **REPE : Repeat while equal**
- **REPZ: Repeat while zero**
- Repeat following CMPSB, CMPSW, SCASB, SCASW instructions while ZF = 1 (result is Equal), maximum CX times.
- While (CX<>0 and ZF=1)
  - *Execute the string instruction*
  - *CX=CX-1*



String operation instructions

- **REPNE : Repeat while not equal**
- **REPNZ : Repeat while not zero**
- Repeat following CMPSB, CMPSW, SCASB, SCASW instructions while ZF = 0 (result is Not Equal), maximum CX times.
- While (CX<>0 and ZF=0)
  - *Execute the string instruction*
  - *CX=CX-1*
-

String operation instructions

- **MOVS / MOVSB / MOVSW – Move string / Move string byte / Move string word**
- Copy byte at DS:[SI] to ES:[DI]. Update SI and DI.

  Algorithm: ES:[DI] = DS:[SI]
  if DF = 0 then
  - *SI = SI + 1 / 2*
  - *DI = DI + 1 / 2*
- else
  - *SI = SI - 1 / 2*
  - *DI = DI - 1 / 2*

Example

```
Mov AX,5000h
Mov DS,AX
Mov AX,6000h
Mov ES,AX
Mov CX,0FFh
Mov SI,1000h
Mov DI,2000h
CLD
REP MOVSB
```

String operation instructions

- **LODS / LODSB / LODSW - Load string / Load string byte / Load string word**
- (LODSB) Load byte at DS:[SI] into AL. Update SI.

  Algorithm:

  AL/AX = DS:[SI]

- if DF = 0 then
  - *SI = SI + 1*
- else
  - *SI = SI - 1*

String operation instructions

- **STOS / STOSB / STOSW – Store string**
- (STOSB) Store byte in AL into ES:[DI]. Update DI.

  Algorithm:

  ES:[DI] = AL/AX

- if DF = 0 then
  - *DI = DI + 1*
- else
  - *DI = DI - 1*

String operation instructions

- **CMPS / CMPSB / CMPSW – Compare string**
- Compare bytes: ES:[DI] from DS:[SI].

  Algorithm:

  DS:[SI] - ES:[DI]

- set flags according to result:
  OF, SF, ZF, AF, PF, CF

- SI , DI changes

Example

```
Mov AX,5000h
Mov DS,AX
Mov AX,6000h
Mov ES,AX
Mov CX,0FFh
Mov SI,1000h
Mov DI,2000h
CLD
REPE  CMPSW
```

String operation instructions

- **SCAS / SCASB / SCASW - Scan string**
- Compare bytes: AL from ES:[DI].

  Algorithm:

  ES:[DI] - AL

- set flags according to result:
  OF, SF, ZF, AF, PF, CF

- DI changes