📘 String Data Transfer Instructions

There are **five main string instructions** in x86 assembly:

1. **LODS** – Load from memory to accumulator
2. **STOS** – Store from accumulator to memory
3. **MOVS** – Move memory to memory
4. **INS** – Input from port to memory
5. **OUTS** – Output from memory to port

Each instruction can work with **byte, word, doubleword, or (in 64-bit) quadword**.

🔷 Direction Flag (D)

- Located in the **flags register**.
- Controls **auto-increment / auto-decrement** for DI and SI during string operations.

Instruction	D = 0	D = 1
Any string instruction	**Increment** SI/DI	**Decrement** SI/DI

- Use `CLD` → clear D flag → auto-increment
- Use `STD` → set D flag → auto-decrement

🔷 DI & SI Registers

- **SI (Source Index)** → accesses memory in **DS** by default
- **DI (Destination Index)** → accesses memory in **ES** by default
- **MOVS** uses both DI & SI → allows moving **64K bytes from one segment to another**
- **32-bit mode** → use **ESI** and **EDI** instead of SI & DI

🔷 LODS – Load String

- Loads **AL/AX/EAX/RAX** with data from memory pointed by **SI/ESI**.
- Auto-adjust SI depending on **D flag** and data size:

Instruction	Operation	SI increment/ decrement
`LODSB`	AL = DS:[SI]	±1
`LODSW`	AX = DS:[SI]	±2
`LODSD`	EAX = DS:[SI]	±4
`LODSQ`	RAX = DS: [RSI]	±8

Example:

```
DS = 1000H, SI = 1000H, D = 0
LODSW ; AX = [1000H:1000H], SI += 2
```

- Only **SI** changes, DI is unaffected.

যদি **DF (Direction Flag) = 0** → SI/ESI/RSI **increment হবে** (আগে থেকে পরে পড়বে)।

যদি **DF = 1** → SI/ESI/RSI **decrement হবে** (পরে থেকে আগে পড়বে)।

LODSB (Load String Byte). LODSW (Load String Word). LODSD (Load String Doubleword) . LODSQ (Load String Quadword) (64-bit mode)

🔷 STOS – Store String-AL/AX/EAX/RAX রেজিস্টারের ভ্যালু মেমোরিতে রাখবে,

আর সেই মেমোরির ঠিকানা **DI / EDI / RDI** রেজিস্টার দিয়ে ঠিক করা হয়।

- Stores **AL/AX/EAX/RAX** at memory pointed by **DI/EDI** in **ES**.
- Auto-adjust DI depending on **D flag** and data size:

Instruction	Operation	DI increment/ decrement
`STOSB`	ES:[DI] = AL	±1
`STOSW`	ES:[DI] = AX	±2
`STOSD`	ES:[DI] = EAX	±4
`STOSQ`	ES:[RDI] = RAX	±8

1. **D flag (Direction Flag)** দেখে নির্ধারণ হয় DI বাড়বে নাকি কমবে।
    - D=0 হলে → DI **ইনক্রিমেন্ট (++)** হবে। D=1 হলে → DI **ডিক্রিমেন্ট (--)** হবে। সাধারণত **DI = গন্তব্য ঠিকানা (Destination Index)**
      অর্থাৎ AL/AX/EAX এর ভ্যালু কোথায় লিখে রাখা হবে সেটা DI ঠিক করে দেয়।

Example: Clear memory buffer using `REP STOSW`:

```
void ClearBuffer(int count, short* buffer)
{
_asm{
    push edi
    push es
    push ds
    mov ax,0          ; value to store
    mov ecx,count     ; number of words
    mov edi,buffer    ; destination
    pop es            ; ES = DS
    rep stosw         ; store AX repeatedly
    pop es
    pop edi
}
}
```

1. **ax = 0** → আমরা চাই প্রতিবার মেমোরিতে 0 লিখতে।
2. **ecx = count** → কতবার লিখবো সেটা গুনে রাখা হলো।
3. **edi = buffer** → কোন জায়গায় লিখবো সেটা ঠিক করা হলো।
4. **rep stosw** → `AX` (মানে 0) প্রতিবার **২ byte (word)** আকারে buffer-এ লিখবে, যতক্ষণ না **ecx=0** হয়।
    - প্রতিবার লেখার পর `edi += 2` (word = 2 byte বলে)।

👉 এর ফলে পুরো buffer **0 দিয়ে ভরে যাবে**। Buffer শুরুতে (garbage data থাকতে পারে):[123, -45, 999, 77, -100]

REP STOSW দিয়ে 0 করলে:[0, 0, 0, 0, 0]

এখন তুমি নিশ্চিত → সবখানে 0, Garbage নেই।
এরপর তুমি নিজের দরকার মতো ভ্যালু লিখতে পারবে।

- `REP` → repeats instruction **CX times**
- DI auto-increments/decrements → fills memory block

🔷 Key Points

1. **LODS** → load **source** to accumulator (SI affected)

2. **STOS** → store **accumulator** to **destination** (DI affected)
3. **MOVS** → copy **memory** → **memory** using SI & DI
4. **INS / OUTS** → input/output with ports
5. **Direction flag (D)** controls whether SI/DI increment or decrement
6. **REP prefix** → repeats instruction multiple times (except LODS)

🔷 MOVS – Memory-to-Memory Transfer

- **Purpose:** Move data from **DS:SI** (source) to **ES:DI** (destination).
- **Only memory-to-memory instruction** in 8086–Pentium4.
- Supports **byte, word, doubleword**, or **quadword (64-bit mode)**.
- Auto-adjust **SI and DI** based on **direction flag D**.

Forms of MOVS

Instruction	Operation	SI/DI increment/ decrement
MOVSB	ES:[DI] = DS:[SI]	±1
MOVSW	ES:[DI] = DS:[SI]	±2
MOVSD	ES:[DI] = DS:[SI]	±4
MOVSQ	[RDI] = [RSI] (64-bit)	±8

- `REP` → repeats MOVSD **CX times**
- SI/DI auto-increment → transfers **entire block** efficiently

🔷 INS – Input String from I/O Device

- Transfers data from **I/O device (DX)** to **ES:DI**.
- **Not available on 8086/8088**.
- Data size: **byte, word, doubleword**.
- Can use **REP prefix** to input a block of data.

Forms of INS

Instruction	Operation	DI increment
INSB	ES:[DI] = [DX]	±1
INSW	ES:[DI] = [DX]	±2
INSD	ES:[DI] = [DX]	±4

Example: Input 50 bytes from I/O device 03ACh → LISTS

```
MOV DI, OFFSET LISTS  ; address array
MOV DX, 03ACh          ; I/O device
CLD                    ; auto-increment
MOV CX, 50             ; counter
REP INSB               ; input 50 bytes
```

🔷 OUTS – Output String to I/O Device

- Transfers data from **DS:SI** to **I/O device (DX)**.
- **Not available on 8086/8088**.
- Data size: **byte, word, doubleword**.
- Can use **REP prefix** to output a block of data.

Forms of OUTS

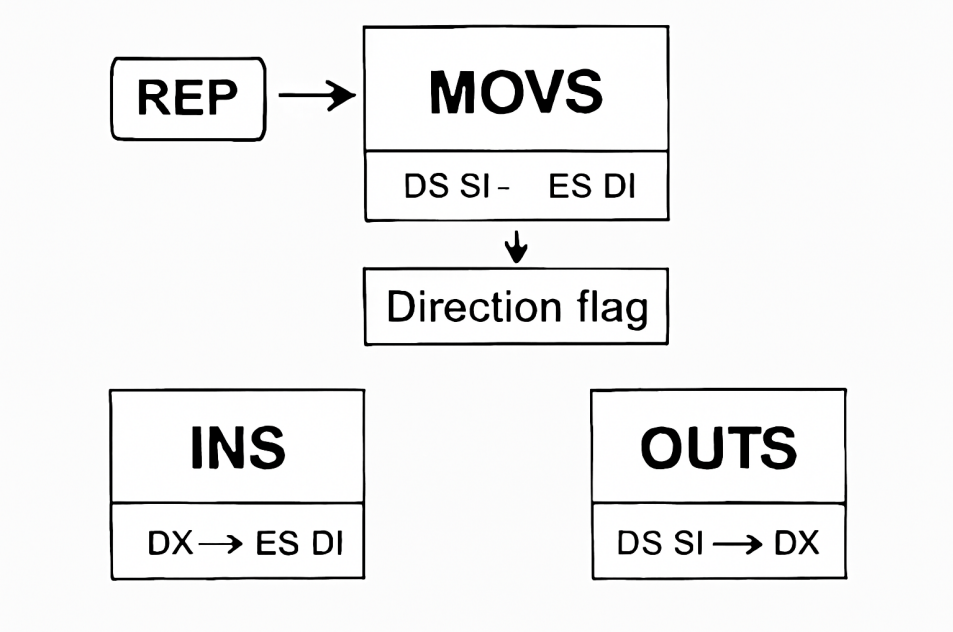
Instruction	Operation	SI increment
OUTSB	[DX] = DS:[SI]	±1
OUTSW	[DX] = DS:[SI]	±2
OUTSD	[DX] = DS:[SI]	±4

Example: Output 100 bytes from ARRAY → I/O device 03ACh

```
MOV SI, OFFSET ARRAY  ; address array
MOV DX, 03ACh         ; I/O device
CLD                   ; auto-increment
MOV CX, 100           ; counter
REP OUTSB             ; output 100 bytes
```

🔷 Key Notes for String Instructions

1. **Direction Flag D** controls increment/decrement of SI/DI.
2. **REP prefix** repeats the instruction **CX/RCX times**.
3. **MOVS** → memory → memory (DS:SI → ES:DI)
4. **INS** → I/O → memory (DX → ES:DI)
5. **OUTS** → memory → I/O (DS:SI → DX)
6. SI → source, DI → destination, auto-adjusted by D flag.



🔷 1. XCHG – Exchange Registers or Register ↔ Memory

- **Purpose:** Swap contents of **two registers** or a **register and memory**.
- Cannot exchange **segment registers** or **memory-to-memory**.
- Sizes: **byte, word, doubleword**, 64-bit in 64-bit mode.
- Most efficient: `XCHG AX, reg16` → only 1 byte instruction.

Examples:

Instruction	Operation
XCHG AL,CL	Swap AL ↔ CL
XCHG CX,BP	Swap CX ↔ BP
XCHG EDX,ESI	Swap EDX ↔ ESI

XCHG AL,DATA2	Swap AL ↔ memory DATA2
XCHG RBX,RCX	Swap RBX ↔ RCX (64-bit)

Note: For memory addressing, order of operands doesn't matter:
`XCHG AL,[DI]` = `XCHG [DI],AL`.

🔷 2. LAHF & SAHF – Load/Store Flags to AH

- **LAHF:** Load **low 8 bits of FLAGS** → AH register
- **SAHF:** Store AH → low 8 bits of FLAGS
- Mainly **legacy**, for translating 8085 code to 8086.
- Rarely used today; sometimes used with **numeric coprocessor** (FPU).
- **Not valid in 64-bit mode.**

🔷 3. XLAT – Table Lookup/Translate

- Converts **AL** into a code from a **lookup table in memory**.
- Formula: `AL = [DS:BX + AL]` → AL now contains table value.
- Useful for **BCD to 7-segment display** conversion.

Example: BCD → 7-segment

```
TABLE DB 3FH, 06H, 5BH, 4FH, 66H, 6DH, 7DH, 27H, 7FH, 6FH
MOV AL, 05        ; AL = 5 (BCD)
MOV BX, OFFSET TABLE
XLAT              ; AL = 6DH (7-segment code)
```

🔷 4. IN & OUT – I/O Data Transfer

- **IN:** Input from I/O port → AL/AX/EAX
- **OUT:** Output AL/AX/EAX → I/O port
- **Port addressing types:**
    1. **Fixed Port:** Port number in instruction (`IN AL,6AH`)
    2. **Variable Port:** Port number in `DX` register (`IN AL,DX`)

Forms of IN/OUT

Instruction	Operation
IN AL,p8	Input 8 bits from port p8 → AL
IN AX,p8	Input 16 bits from port p8 → AX
IN EAX,p8	Input 32 bits from port p8 → EAX
IN AL,DX	Input 8 bits from port DX → AL
OUT p8,AL	Output 8 bits from AL → port p8
OUT DX,AX	Output 16 bits from AX → port DX

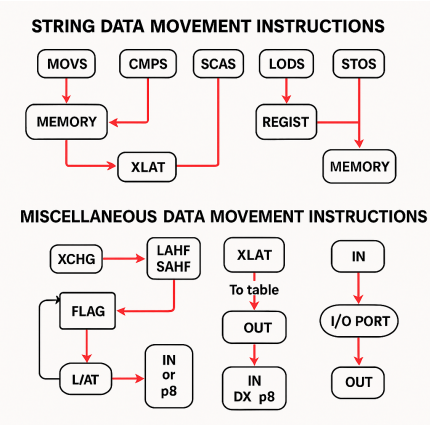
Example: Click PC speaker (port 61H)

```
IN AL,61H        ; read current port
OR AL,03H        ; set rightmost 2 bits = 11
OUT 61H,AL       ; write to port
AND AL,FC        ; clear rightmost 2 bits = 00
OUT 61H,AL       ; write to port
MOV CX,8000H     ; delay
LOOP L1
```

- Adjust `CX` for **duration of click**.
- Repeat program for **series of clicks**.

🔷 Key Points

1. **XCHG:** Efficient swap, supports memory only with one register.
2. **LAHF/SAHF:** Legacy flags transfer, not in 64-bit mode.
3. **XLAT:** Table lookup; AL → memory table value.
4. **IN/OUT:** Input/output from I/O ports; supports fixed or variable port.
5. Variable port = use DX, fixed port = port in instruction.



🔷 5. MOVSX – Move and Sign-Extend

- **Purpose:** Move a smaller signed value (8- or 16-bit) into a larger register and **extend the sign bit**.
- **Sign-extension:** Copies the **most significant bit (MSB)** of the source into the extended part.

- Used for **signed numbers**.

Examples:

Instruction	Operation
MOVSX CX,BL	Sign-extend 8-bit BL → 16-bit CX
MOVSX ECX,AX	Sign-extend 16-bit AX → 32-bit ECX
MOVSX BX,DATA1	Sign-extend byte at DATA1 → BX
MOVSX EAX,[EDI]	Sign-extend word at memory EDI → EAX
MOVSX RAX, [RDI]	Sign-extend doubleword at memory RDI → RAX (64-bit mode)

Example:
If BL = 84H → 1000 0100_2 → sign-extended to CX → FFFF 0084H (16-bit)

🔷 6. MOVZX – Move and Zero-Extend

- **Purpose:** Move a smaller value (8- or 16-bit) into a larger register and **fill upper bits with 0**.
- **Zero-extension:** MSBs filled with 0.
- Used for **unsigned numbers**.

Examples:

Instruction	Operation
MOVZX DX,AL	Zero-extend AL → DX
MOVZX EBP,DI	Zero-extend DI → EBP
MOVZX DX,DATA2	Zero-extend byte at DATA2 → DX
MOVZX EAX,DATA3	Zero-extend word at DATA3 → EAX
MOVZX RBX,ECX	Zero-extend ECX → RBX

Example:
AL = 34H → 0011 0100_2 → zero-extended to DX → 0034H

🔷 7. BSWAP – Byte Swap (80486+)

- **Purpose:** Swap bytes of a 32-bit register.
- Reverses **big-endian ↔ little-endian** formats.

- **Little Endian (Intel CPU-তে default)সবচেয়ে ছোট byte (LSB)** → মেমোরির **সবচেয়ে ছোট address** এ রাখা হয়।**সবচেয়ে বড় byte (MSB)** → মেমোরির **সবচেয়ে বড় address** এ রাখা হয়।**Big Endian-সবচেয়ে বড় byte (MSB)** → মেমোরির **সবচেয়ে ছোট address** এ রাখা হয় ,**সবচেয়ে ছোট byte (LSB)** → মেমোরির **সবচেয়ে বড় address** এ রাখা হয়।
- **64-bit mode:** swaps all 8 bytes in a 64-bit register.

Example:

```
EAX = 00112233H
BSWAP EAX
; Result → EAX = 33221100H
```

- Swaps: 1st ↔ 4th byte, 2nd ↔ 3rd byte.

🔷 8. CMOV – Conditional Move (Pentium Pro+)

- **Purpose:** Move **only if a condition is true**.
- **Source:** 16- or 32-bit register/memory
- **Destination:** 16- or 32-bit register
- Requires **.686 switch** in assembler.

Conditions & Instructions:

Instruction	Flag Tested	Operation
CMOVB	C = 1	Move if below
CMOVAE	C = 0	Move if above or equal
CMOVBE	Z = 1 or C = 1	Move if below or equal
CMOVA	Z = 0 and C = 0	Move if above
CMOVE / CMOVZ	Z = 1	Move if equal / zero
CMOVNE / CMOVNZ	Z = 0	Move if not equal / not zero
CMOVL	S ≠ O	Move if less than
CMOVLE	Z = 1 or S ≠ O	Move if less than or equal
CMOVG	Z = 0 and S = O	Move if greater than
CMOVGE	S = O	Move if greater or equal
CMOVS	S = 1	Move if negative
CMOVNS	S = 0	Move if positive
CMOVC	C = 1	Move if carry

CMOVNC	C = 0	Move if no carry
CMOVO	O = 1	Move if overflow
CMOVNO	O = 0	Move if no overflow
CMOVP / CMOVPE	P = 1	Move if parity even
CMOVNP / CMOVPO	P = 0	Move if parity odd

Use-case: Avoid branches; useful for optimizing **if-then statements**.

🔷 Summary Diagram (Memory / Register / I/O)

Instruction	Source	Destination	Notes
MOVSX	Mem/Reg	Reg	Sign-extend
MOVZX	Mem/Reg	Reg	Zero-extend
BSWAP	Reg	Reg	Reverse bytes
CMOV	Reg/Mem	Reg	Conditional move
IN	Port	AL/AX/EAX	Input from I/O port
OUT	AL/AX/EAX	Port	Output to I/O port
XLAT	Mem[AL+BX]	AL	Table lookup
XCHG	Reg/Reg or Reg/Mem	Reg	Exchange contents
LAHF	FLAGS	AH	Legacy
SAHF	AH	FLAGS	Legacy