Weeks 6

8088/8086 Microprocessor Programming
Shift

- **SHL**
- **SAL**
- **SHR**
- **SAR**

Target register or memory

- **C**
- **0**
- **C**
- **C**

Sign Bit

equivalent

2
Examples

Examples

SHL AX, 1
SAL DATA1, CL ; shift count is a modulo-32 count

Ex. ; Multiply AX by 10
    SHL AX, 1
    MOV BX, AX
    MOV CL, 2
    SHL AX, CL
    ADD AX, BX

Ex. What are the results of SAR CL, 1 if CL initially contains B6H?

Ex. What are the results of SHL AL, CL if AL contains 75H and CL contains 3?
What is the result of ROL byte ptr [SI], 1 if this memory location 3C020 contains 41H?

Ex.
What is the result of ROL word ptr [SI], 8 if this memory location 3C020 contains 4125H?
Example

Write a program that counts the number of 1’s in a byte and writes it into BL

DATA1   DB 97        ; 61h
SUB      BL,BL       ; clear BL to keep the number of 1s
MOV      DL,8        ; rotate total of 8 times
MOV      AL,DATA1
AGAIN:   ROL         AL,1       ; rotate it once
         JNC        NEXT       ; check for 1
         INC        BL         ; if CF=1 then add one to count
NEXT:    DEC         DL         ; go through this 8 times
         JNZ        AGAIN      ; if not finished go back
         NOP
BCD and ASCII Numbers

• BCD (Binary Coded Decimal)
  – Unpacked BCD: One byte per digit
  – Packed BCD: 4 bits per digit (more efficient in storing data)

• ASCII to unpacked BCD conversion
  – Keyboards, printers, and monitors all use ASCII.
  – Digits 0 to 9 are represented by ASCII codes 30 – 39.

• Example. Write an 8086 program that displays the packed BCD number in register AL on the system video monitor
  – The first number to be displayed should be the MS Nibble
  – It is found by masking the LS Nibble and then rotating the MS Nibble into the LSD position
  – The result is then converted to ASCII by adding 30h
  – The BIOS video service is then called to display this result.
ASCII Numbers Example

MOV BL, AL; save
AND AL, F0H
MOV CL, 4
ROR AL, CL
ADD AL, 30H
MOV AH, 0EH
INT 10H ; display single character

MOV AL, BL; use again
AND AL, 0FH
ADD AL, 30H
INT 10H
INT 20H ; RETURN TO DOS
Example

• Write an 8086 program that adds two packed BCD numbers input from the keyboard and computes and displays the result on the system video monitor.
• Data should be in the form 64+89= The answer 153 should appear in the next line.
Example Continued

Mov dx, offset bufferaddress
Mov ah, 0a
Mov si, dx
Mov byte ptr [si], 6
Int 21
Mov ah, 0eh
Mov al, 0ah
Int 10
; BIOS service 0e line feed position cursor

sub byte ptr [si+2], 30h
sub byte ptr [si+3], 30h
sub byte ptr [si+5], 30h
sub byte ptr [si+6], 30h

Mov cl, 4
Rol byte ptr [si+3], cl
Rol byte ptr [si+6], cl
Ror word ptr [si+5], cl
Ror word ptr [si+2], cl
Mov al, [si+3]
Add al, [si+6]
Daa
Mov bh, al
Jnc display
Mov al, 1
Call display
Mov al, bh
Call display
Int 20

6 | ? | 6 | 4 | + | 8 | 9 | =

0 | 1 | 2 | 3 | 4 | 5 | 6 | 7

9
Flag Control Instructions

<table>
<thead>
<tr>
<th>SF</th>
<th>ZF</th>
<th>AF</th>
<th>PF</th>
<th>CF</th>
</tr>
</thead>
</table>

- **LAHF** Load AH from flags (AH) ← (Flags)
- **SAHF** Store AH into flags (Flags) ← (AH)
  - Flags affected: SF, ZF, AF, PF, CF
- **CLC** Clear Carry Flag (CF) ← 0
- **STC** Set Carry Flag (CF) ← 1
- **CLI** Clear Interrupt Flag (IF) ← 0
- **STI** Set interrupt flag (IF) ← 1
- **Example (try with debug)**
  LAHF
  MOV AX,0000
  ADD AX,00
  SAHF
  - Check the flag changes!

Bulk manipulation of the flags

Individual manipulation of the flags
## Compare

### Unsigned Comparison

<table>
<thead>
<tr>
<th>Comp Operands</th>
<th>CF</th>
<th>ZF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dest &gt; source</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Dest = source</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Dest &lt; source</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

### Signed Comparison

<table>
<thead>
<tr>
<th>Comp Operands</th>
<th>ZF</th>
<th>SF,OF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dest &gt; source</td>
<td>0</td>
<td>SF=OF</td>
</tr>
<tr>
<td>Dest = source</td>
<td>1</td>
<td>x</td>
</tr>
<tr>
<td>Dest &lt; source</td>
<td>0</td>
<td>SF&lt;&gt;OF</td>
</tr>
</tbody>
</table>

### Mnemonic Table

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Meaning</th>
<th>Format</th>
<th>Operation</th>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMP</td>
<td>Compare</td>
<td>CMP D,S</td>
<td>(D) - (S) is used in setting or resetting the flags</td>
<td>CF, AF, OF, PF, SF, ZF</td>
</tr>
</tbody>
</table>

(a)
Compare Example

DATA1 DW 235Fh
...
MOV AX, CCCCH
CMP AX, DATA1
JNC OVER
SUB AX, AX
OVER: INC DATA1

CCCC – 235F = A96D => Z=0, CF=0 =>
CCCC > DATA1
Write a program to find the **highest** among 5 grades and write it in **DL**

<table>
<thead>
<tr>
<th>DATA</th>
<th>DB</th>
<th>51, 44, 99, 88, 80</th>
<th>;13h,2ch,63h,58h,50h</th>
<th>MOV CX, 5</th>
<th>;set up loop counter</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV</td>
<td>BX, OFFSET DATA</td>
<td>;BX points to GRADE data</td>
<td>SUB AL, AL</td>
<td>;AL holds highest grade found so far</td>
<td></td>
</tr>
<tr>
<td>AGAIN:</td>
<td>CMP AL, [BX]</td>
<td>;compare next grade to highest</td>
<td>JA NEXT</td>
<td>;jump if AL still highest</td>
<td></td>
</tr>
<tr>
<td>NEXT:</td>
<td>INC BX</td>
<td>;point to next grade</td>
<td>LOOP AGAIN</td>
<td>;continue search</td>
<td></td>
</tr>
<tr>
<td>MOV</td>
<td>DL, AL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Jump Instructions

- Unconditional vs conditional jump
## Conditional Jump

These flags are based on general comparison

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Flags/Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>JZ</td>
<td>Jump if ZERO</td>
<td>ZF = 1</td>
</tr>
<tr>
<td>JE</td>
<td>Jump if EQUAL</td>
<td>ZF = 1</td>
</tr>
<tr>
<td>JNZ</td>
<td>Jump if NOT ZERO</td>
<td>ZF = 0</td>
</tr>
<tr>
<td>JNE</td>
<td>Jump if NOT EQUAL</td>
<td>ZF = 0</td>
</tr>
<tr>
<td>JC</td>
<td>Jump if CARRY</td>
<td>CF = 1</td>
</tr>
<tr>
<td>JNC</td>
<td>Jump if NO CARRY</td>
<td>CF = 0</td>
</tr>
<tr>
<td>JCXZ</td>
<td>Jump if CX = 0</td>
<td>CX = 0</td>
</tr>
<tr>
<td>JECXZ</td>
<td>Jump if ECX = 0</td>
<td>ECX = 0</td>
</tr>
</tbody>
</table>
## Conditional Jump based on flags

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Flags/Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>JS</td>
<td>JUMP IF SIGN (NEGATIVE)</td>
<td>SF = 1</td>
</tr>
<tr>
<td>JNS</td>
<td>JUMP IF NOT SIGN (POSITIVE)</td>
<td>SF = 0</td>
</tr>
<tr>
<td>JP</td>
<td>Jump if PARITY EVEN</td>
<td>PF = 1</td>
</tr>
<tr>
<td>JNP</td>
<td>Jump if PARITY ODD</td>
<td>PF = 0</td>
</tr>
<tr>
<td>JO</td>
<td>JUMP IF OVERFLOW</td>
<td>OF = 1</td>
</tr>
<tr>
<td>JNO</td>
<td>JUMP IF NO OVERFLOW</td>
<td>OF = 0</td>
</tr>
</tbody>
</table>
# Jump Based on Unsigned Comparison

These flags are based on unsigned comparison

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Flags/Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>JA</td>
<td>Jump if above op1&gt;op2</td>
<td>CF = 0 and ZF = 0</td>
</tr>
<tr>
<td>JNBE</td>
<td>Jump if not below or equal</td>
<td>CF = 0 and ZF = 0</td>
</tr>
<tr>
<td></td>
<td>op1 not &lt;= op2</td>
<td></td>
</tr>
<tr>
<td>JAE</td>
<td>Jump if above or equal</td>
<td>CF = 0</td>
</tr>
<tr>
<td></td>
<td>op1&gt;=op2</td>
<td></td>
</tr>
<tr>
<td>JNB</td>
<td>Jump if not below op1 not &lt;</td>
<td>CF = 0</td>
</tr>
<tr>
<td></td>
<td>op2</td>
<td></td>
</tr>
<tr>
<td>JB</td>
<td>Jump if below op1&lt;op2</td>
<td>CF = 1</td>
</tr>
<tr>
<td>JNAE</td>
<td>Jump if not above nor equal</td>
<td>CF = 1</td>
</tr>
<tr>
<td></td>
<td>op1&lt;op2</td>
<td></td>
</tr>
<tr>
<td>JBE</td>
<td>Jump if below or equal</td>
<td>CF = 1 or ZF = 1</td>
</tr>
<tr>
<td></td>
<td>op1 &lt;= op2</td>
<td></td>
</tr>
<tr>
<td>JNA</td>
<td>Jump if not above op1 &lt;= op2</td>
<td>CF = 1 or ZF = 1</td>
</tr>
</tbody>
</table>
# Jump Based on Signed Comparison

These flags are based on signed comparison

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Flags/Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>JG</td>
<td>Jump if GREATER op1&gt;op2</td>
<td>SF = OF AND ZF = 0</td>
</tr>
<tr>
<td>JNLE</td>
<td>Jump if not LESS THAN or equal op1&gt;op2</td>
<td>SF = OF AND ZF = 0</td>
</tr>
<tr>
<td>JGE</td>
<td>Jump if GREATER THAN or equal op1&gt;=op2</td>
<td>SF = OF</td>
</tr>
<tr>
<td>JNL</td>
<td>Jump if not LESS THAN op1&gt;=op2</td>
<td>SF = OF</td>
</tr>
<tr>
<td>JL</td>
<td>Jump if LESS THAN op1&lt;op2</td>
<td>SF &lt;&gt; OF</td>
</tr>
<tr>
<td>JNGE</td>
<td>Jump if not GREATER THAN nor equal op1&lt;op2</td>
<td>SF &lt;&gt; OF</td>
</tr>
<tr>
<td>JLE</td>
<td>Jump if LESS THAN or equal op1 &lt;= op2</td>
<td>ZF = 1 OR SF &lt;&gt; OF</td>
</tr>
<tr>
<td>JNG</td>
<td>Jump if NOT GREATER THAN op1 &lt;= op2</td>
<td>ZF = 1 OR SF &lt;&gt; OF</td>
</tr>
</tbody>
</table>
Control Transfer Instructions (conditional)

- It is often necessary to transfer the program execution.
  - Short
    - A special form of the direct jump: “short jump”
    - All conditional jumps are short jumps
    - Used whenever target address is in range +127 or −128 (single byte)
    - Instead of specifying the address a relative offset is used.
Short Jumps

• Conditional Jump is a **two byte instruction**.

• In a jump backward the second byte is the 2’s complement of the displacement value.

• To calculate the target the second byte is added to the IP of the instruction after the jump.

Ex:

000D ADD AL,[BX]
000F INC BX
0010 DEC CX
0011 JNZ FA
0013

Short Jump 0013 + FA (-6) = 0D
.model small
.stack 100h
.data
org 0010
message1 db "You now have a small letter entered !",0dh,0ah,'$'
org 50
message2 db "You have NON small letters ",0dh,0ah,'$'
.code
main proc
    mov ax,@data
    mov ds,ax
    mov ah,00h
    int 16h
    cmp al,61h
    jb next
    cmp al,7Ah
    ja next
    mov ah,09h
    mov dx,offset message1
    int 21h
    int 20h
next: mov dx,offset message2
    mov ah,09h
    int 21h
    int 20h
    mov ax,4C00h
    int 21h
main endp
end main
A Simple Example Program finds the sum

- Write a program that adds 5 bytes of data and saves the result. The data should be the following numbers: 25, 12, 15, 10, 11

```
.model small
.stack 100h
.data
    Data_in DB 25,12,15,10,11
    Sum   DB  ?
.code
main proc far
    mov ax, @Data
    mov ds,ax
    mov cx,05h
    mov bx,offset data_in
    mov al,0
    Again: add al,[bx]
        inc bx
        dec cx
        jnz Again
    mov sum,al
    mov ah,4Ch
    INT 21H
Main endp
end main
```
Example Output
Unconditional Jump

- **Short Jump**: `jmp short L1` (8 bit)

- **Near Jump**: `jmp near ptr Label`
  
  If the control is transferred to a memory location within the current code segment (intrasegment), it is NEAR. IP is updated and CS remains the same.

  ➢ The displacement (16 bit) is added to the IP of the instruction following jump instruction. The displacement can be in the range of –32,768 to 32,768.

  ➢ The target address can be register indirect, or assigned by the label.

  ➢ **Register indirect JMP**: the target address is the contents of two memory locations pointed at by the register.

  ➢ Ex: `JMP [SI]` will replace the IP with the contents of the memory locations pointed by DS:DI and DS:DI+1 or `JMP [BP + SI + 1000]` in SS.

- **Far Jump**: If the control is transferred to a memory location outside the current segment. Control is passing outside the current segment both CS and IP have to be updated to the new values. ex: `JMP FAR PTR label = EA 00 10 00 20 jmp far ptr Label` ; this is a jump out of the current segment.
Near Jump

Jumps to the specified IP with +/- 32K distance from the next instruction following the jmp instruction
## Far Jump

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Memory Address</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0B20:1000</td>
<td><code>jmp 3000:1200</code></td>
<td></td>
<td></td>
<td>Jumps to the specified CS:IP</td>
</tr>
<tr>
<td>0B20:1005</td>
<td><code>-u 1000</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0B20:1000</td>
<td><code>EA00120030</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0B20:1005</td>
<td><code>FF750B</code></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**XLAT**

- Adds the contents of AL to BX and uses the resulting offset to point to an entry in an 8 bit translate table.
- This table contains values that are substituted for the original value in AL.
- The byte in the table entry pointed to by BX+AL is moved to AL.

- XLAT [tablename] ; optional because table is assumed at BX

- Table db ‘0123456789ABCDEF’

Mov AL,0A; index value
Mov bx,offset table
Xlat; AL=41h, or ‘A’
A subroutine is a special segment of a program that can be called for execution from any point in the program.

A RET instruction must be included at the end of the subroutine to initiate the return sequence to the main program environment.

Examples. Call 1234h  
Call BX  
Call [BX]

Two calls
• intrasegment
• intersegment
Calling a NEAR proc

- The CALL instruction and the subroutine it calls are in the same segment.
- Save the current value of the IP on the stack.
- Load the subroutine’s offset into IP (nextinst + offset)

<table>
<thead>
<tr>
<th>Calling Program</th>
<th>Subroutine</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main proc</td>
<td>sub1 proc</td>
<td></td>
</tr>
<tr>
<td>001A: call sub1</td>
<td>0080: mov ax,1</td>
<td></td>
</tr>
<tr>
<td>001D: inc ax</td>
<td>...</td>
<td>1fffd</td>
</tr>
<tr>
<td>.</td>
<td>ret</td>
<td>1D</td>
</tr>
<tr>
<td>Main endp</td>
<td>sub1 endp</td>
<td>1ffe</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1fff</td>
</tr>
</tbody>
</table>
The CALL instruction and the subroutine it calls are in the “Different” segments.

Save the current value of the CS and IP on the stack.

Then load the subroutine’s CS and offset into IP.

**Calling Program** | **Subroutine** | **Stack**
---|---|---
Main **proc** | sub1 **proc far** |
1FCB:001A: call far ptr sub1 | 4EFA:0080: mov ax,1 |
1FCB:001F: inc ax | .... |
... | .... |
... | ret (retf opcode generated) |
Main **endp** | sub1 **endp** |

 Opcode 8000  FA4E
Example on Far/Near Procedure Calls

0350:1C00 Call FarProc
0350:1C05 Call NearProc
0350:1C08 nop
Nested Procedure Calls

A subroutine may itself call other subroutines.

Example:

```
main proc
  000A call subr1
  000C mov ax,…
  …
  main endp

subr1 proc
  0030 nop
  …
  call subr2
  0040 ret …
  subr1 endp

subr2 proc
  0050 nop
  …
  call subr3
  0060 ret …
  subr2 endp

subr3 proc
  0070 nop
  …
  0079 nop
  007A ret
  subr3 endp
```

Q: show the stack contents at 0079?

```
1ff0   60
1ffa   00
1ffb   40
1ffc   00
1ffd   0c
1ffe   00
1fff   X
```
## Push and Pop Instructions

<table>
<thead>
<tr>
<th>To save registers and parameters on the stack</th>
<th>Push S (16/32 bit or Mem)</th>
<th>((SP) \leftarrow (SP) - 2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>{</td>
<td>{</td>
<td>((SP)) \leftarrow (S)</td>
</tr>
<tr>
<td>{ PUSH (XX) }</td>
<td>{</td>
<td></td>
</tr>
<tr>
<td>{ PUSH (YY) }</td>
<td>{</td>
<td></td>
</tr>
<tr>
<td>{ PUSH (ZZ) }</td>
<td>{</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Main body of the subroutine</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>{</td>
<td>{</td>
<td></td>
</tr>
<tr>
<td>{</td>
<td>{</td>
<td></td>
</tr>
<tr>
<td>{</td>
<td>{</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>To restore registers and parameters from the stack</th>
<th>Pop D (16/32 bit or Mem)</th>
<th>((D) \leftarrow ((SP)))</th>
</tr>
</thead>
<tbody>
<tr>
<td>{</td>
<td>{</td>
<td>((SP) \leftarrow (SP) + 2)</td>
</tr>
<tr>
<td>{ POP (ZZ) }</td>
<td>{</td>
<td></td>
</tr>
<tr>
<td>{ POP (YY) }</td>
<td>{</td>
<td></td>
</tr>
<tr>
<td>{ POP (XX) }</td>
<td>{</td>
<td></td>
</tr>
<tr>
<td>{ RET }</td>
<td>{</td>
<td></td>
</tr>
</tbody>
</table>
# Loop and Loop Handling Instructions

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Meaning</th>
<th>Format</th>
<th>Operation</th>
</tr>
</thead>
</table>
| LOOP              | Loop                           | LOOP Short-label        | (CX) ← (CX) - 1  
Jump is initiated to location defined by short-label if (CX) ≠ 0; otherwise, execute next sequential instruction |
| LOOPE/LOOPZ       | Loop while equal/loop while zero | LOOPE/LOOPZ Short-label | (CX) ← (CX) - 1  
Jump to location defined by short-label if (CX) ≠ 0 and (ZF) = 1; otherwise, execute next sequential instruction |
| LOOPNE/LOOPNZ     | Loop while not equal/loop while not zero | LOOPNE/LOOPNZ Short-label | (CX) ← (CX) - 1  
Jump to location defined by short-label if (CX) ≠ 0 and (ZF) = 0; otherwise, execute next sequential instruction |

*Figure 6-28  Loop instructions.*
Loop

NEXT:

**MOV CX,COUNT**

Load count for the number of repeats

... Body of routine that is repeated ...

**LOOP** NEXT

Loop back to label NEXT if count not zero

(a)

MOV AX,DATASEGADDR
MOV DS,AX
MOV SI,BLK1ADDR
MOV DI,BLK2ADDR
MOV CX,N

NXTPT:

MOV AH,[SI]
MOV [DI],AH
INC SI
INC DI
LOOP NXTPT
HLT

(b)
Nested Loops

**single Loop**

MOV CX,A
BACK: ...
...
...
LOOP BACK

**Nested Loops**

MOV CX,A
OUTER: PUSH CX
MOV CX, 99
INNER: NOP
...
...
...
LOOP INNER
POP CX
LOOP OUTER

How many times will the loop execute, if JCXZ wasn’t there?

MOV CX,0
DLOOP: JCXZ SKIP ;guarding
BACK: MUL AX,2H
ADD AX,05H
LOOP BACK
SKIP: INC AX; if CX=0
INT operates similar to Call

- Processor first pushes the flags
- Trace Flag and Interrupt-enable flags are cleared
- Next the processor pushes the current CS register onto the stack
- Next the IP register is pushed

Example: What is the sequence of events for INT 08? If it generates a CS:IP of 0100:0200. The flag is 0081H.
IRET

• IRET must be used for special handling of the stack.
• Must be used at the end of an ISR

<table>
<thead>
<tr>
<th>SP</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SP-6</td>
<td>00</td>
</tr>
<tr>
<td>SP-5</td>
<td>02</td>
</tr>
<tr>
<td>SP-4</td>
<td>00</td>
</tr>
<tr>
<td>SP-3</td>
<td>01</td>
</tr>
<tr>
<td>SP-2</td>
<td>81</td>
</tr>
<tr>
<td>SP-1</td>
<td>00</td>
</tr>
</tbody>
</table>

Return address + flags are loaded

SP initial
80x86 is equipped with special instructions to handle string operations

**String**: A series of data words (or bytes) that reside in consecutive memory locations

Operations: move, scan, compare

**String Instruction:**
- Byte transfer, SI or DI increment or decrement by 1
- Word transfer, SI or DI increment or decrement by 2
- DWord transfer, SI or DI increment or decrement by 4
String Instructions - D Flag

The Direction Flag: Selects the **auto increment D=0** or the **auto decrement D=1** operation for the DI and SI registers during string operations. D is used only with strings.

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Meaning</th>
<th>Format</th>
<th>Operation</th>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLD</td>
<td>Clear DF</td>
<td>CLD</td>
<td>(DF) ← 0</td>
<td>DF</td>
</tr>
<tr>
<td>STD</td>
<td>Set DF</td>
<td>STD</td>
<td>(DF) ← 1</td>
<td>DF</td>
</tr>
</tbody>
</table>

CLD $\rightarrow$ Clears the D flag / STD $\rightarrow$ Sets the D flag
## String Instructions

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Meaning</th>
<th>Format</th>
<th>Operation</th>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVNS</td>
<td>Move string</td>
<td>MOVSB/MOVSW</td>
<td>(((ES)0 + (DI)) \rightarrow ((DS)0 + (SI))) ((SI) \rightarrow (SI) \pm 1 \text{ or } 2) ((DI) \rightarrow (DI) \pm 1 \text{ or } 2)</td>
<td>None</td>
</tr>
<tr>
<td>CMPS</td>
<td>Compare string</td>
<td>CMPSB/CMPSW</td>
<td>Set flags as per (((DS)0 + (SI)) - (ES)0 + (DI))) ((SI) \rightarrow (SI) \pm 1 \text{ or } 2) ((DI) \rightarrow (DI) \pm 1 \text{ or } 2)</td>
<td>CF, PF, AF, ZF, SF, OF</td>
</tr>
<tr>
<td>SCAS</td>
<td>Scan string</td>
<td>SCASB/SCASW</td>
<td>Set flags as per ((AL \text{ or } AX) - (ES)0 + (DI))) ((DI) \rightarrow (DI) \pm 1 \text{ or } 2)</td>
<td>CF, PF, AF, ZF, SF, OF</td>
</tr>
<tr>
<td>LODS</td>
<td>Load string</td>
<td>LODSB/LODSW</td>
<td>((AL \text{ or } AX) \rightarrow ((DS)0 + (SI))) ((SI) \rightarrow (SI) \pm 1 \text{ or } 2)</td>
<td>None</td>
</tr>
<tr>
<td>STOS</td>
<td>Store string</td>
<td>STOSB/STOSW</td>
<td>(((ES)0 + (DI)) \rightarrow (AL \text{ or } AX) \pm 1 \text{ or } 2) ((DI) \rightarrow (DI) \pm 1 \text{ or } 2)</td>
<td>None</td>
</tr>
</tbody>
</table>

MOV AX,DATA SEG ADDR  
MOV DS, AX  
MOV ES, AX  
MOV SI, BLK1 ADDR  
MOV DI, BLK2 ADDR  
MOV CX, N  
CLD  
NXTPT: MOVSB  
LOOP NXTPT  
HLT
Repeat String REP

Basic string operations must be repeated in order to process arrays of data; this is done by inserting a repeat prefix.

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Used with:</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>REP</td>
<td>MOVS, STOS</td>
<td>Repeat while not end of string ( CX \neq 0 )</td>
</tr>
<tr>
<td>REPE/REPZ</td>
<td>CMPS, SCAS</td>
<td>Repeat while not end of string and strings are equal ( CX \neq 0 ) and ( ZF = 1 )</td>
</tr>
<tr>
<td>REPNE/REPNZ</td>
<td>CMPS, SCAS</td>
<td>Repeat while not end of string and strings are not equal ( CX \neq 0 ) and ( ZF = 0 )</td>
</tr>
</tbody>
</table>

Figure 6–36  Prefixes for use with the basic string operations.
Example. Find and replace

- Write a program that scans the name “Mr.Gohns” and replaces the “G” with the letter “J”.

```assembly
Datal db  'Mr.Gones','$'
.code
mov es,ds
cld ;set auto increment bit D=0
mov di, offset data1
mov cx,09; number of chars to be scanned
mov al,'G'; char to be compared against
repne SCASB; start scan AL =? ES[DI]
jne Over; if Z=0
dec di; Z=1
mov byte ptr[di], 'J'
Over:  mov ah,09
       mov dx,offset data1
       int 21h; display the resulting String
```
Strings into Video Buffer

Fill the Video Screen with a value

CLD
MOV AX, 0B800H
MOV ES, AX
MOV DI, 0
MOV CX, 2000H
MOV AL, 20h
REP STOSW
Example. Display the ROM BIOS Date

• Write an 8086 program that searches the BIOS ROM for its creation date and displays that date on the monitor.
• If a date cannot be found display the message “date not found”
• Typically the BIOS ROM date is stored in the form xx/xx/xx beginning at system address F000:FFF5
• Each character is in ASCII form and the entire string is terminated with the null character (00)
• Add a ‘$’ character to the end of the string and make it ready for DOS function 09, INT 21