# Seminar 4

Rechnerarchitektur

# Stringbefehle

- Stringbefehle zur Datenübertragung: LODS_ , STOS_ , MOVS_
- Stringbefehle zur Datenzugriff und Vergleich: SCAS_ , CMPS_

- Load String = LODS
- Store String = STOS
- Move String = MOVS
- Scan String = SCAS
- Compare String = CMPS

# Stringbefehle

- Alle Stringbefehle (MOVS_, LODS_, STOS_, CMPS_, SCAS_) haben folgende Gemeinsamkeiten:
  - Die Adressierung erfolgt immer über die Registerpaare DS:ESI und|oder ES:EDI
  - Die beteiligten Indexregister EDI und/oder ESI werden nach der Ausführung des Befehls automatisch verändert. Diese Veränderung hängt von der Bitbreite des Befehls und der Stellung von DF ab:

| | **DF = 0** | **DF = 1** |
|---|---|---|
| Byteoperation | +1 | -1 |
| Wortoperation | +2 | -2 |
| Doppelwortoperation | +4 | -4 |

# Zeichenfolgen (strings of bytes)

- Eine Zeichenfolge ist durch folgende Eigenschaften charakterisiert:
  - **Datentyp** der Elemente – gegeben von der letzten Buchstabe des Befehl Namens (B=Byte, W=Word, D=Doubleword)
  - Die **Adresse des ersten Elementes** aus der Folge:
    - In **DS:ESI** für den Quellestring
    - In **ES:EDI** für den Zielstring
  - Die **Anzahl der Elemente** – angegeben durch **ECX**
  - Die **Richtung** in welcher die Folge durchlaufen wird – gegeben durch **DF**:
    - DF = 0 – kleine Speicheradresse zu großer Speicheradresse
    - DF = 1 – große Speicheradresse zu kleiner Speicheradresse

# LODSB, LODSW, LODSD

- **LODS_**
  - Kopiert ein Byte/Wort/Doppelwort von der Adresse <DS:ESI> nach AL/AX/EAX
  - Falls **DF=0** dann wird **ESI mit Anzahl von gelesenen Bytes inkrementiert**
  - Falls **DF=1** dann wird **ESI mit Anzahl von gelesenen Bytes dekrementiert**

- Beispiel:

```
titel DB 'Beispiele zum Assemblerskript'
...
MOV ESI, titel  ;  Source index = Quellzeiger
;  ''titel'' liegt im Datensegment, DS:ESI zeigt jetzt auf ''titel''
CLD    ; DF = 0, -> Inkrement
LODSB  ;  load string byte ; -> AL='B'
```

# STOSB, STOSW, STOSD

- **STOS_**
  - Speichert das Byte/Wort/Doppelwort in AL/AX/EAX nach das Byte/Wort/Doppelwort von der Adresse <ES:EDI>
  - Falls **DF=0** dann wird **EDI mit Anzahl von geschriebenen Bytes inkrementiert**
  - Falls **DF=1** dann wird **EDI mit Anzahl von geschriebenen Bytes dekrementiert**

# MOVSB, MOVSW, MOSD

- **MOVS_**
  - Kopiert ein Byte/Wort/Doppelwort von der Adresse <DS:ESI> nach <ES:EDI>
  - Falls **DF=0** dann werden **ESI und EDI mit Anzahl von gelesenen Bytes inkrementiert**
  - Falls **DF=1** dann werden **ESI und EDI mit Anzahl von gelesenen Bytes dekrementiert**

# Beispiel

```
mov ECX, string_dim      ;Anzahl Elemente in dem String
mov ESI, source_string   ;speichert den Offset des Strings in ESI
mov EDI, dest_string
CLD
Again:
     LODSB
     STOSB
LOOP Again
```

# Beispiel

```
Again:
      MOVSB
LOOP Again
```

- Da LODS + STOS = MOVS, ist es äquivalent mit:

```
Again:
      LODSB
      STOSB
LOOP Again
```

# Wiederholung

- ***CMP ziel, quelle***
  - Vergleicht die zwei Operanden und setzt die Flags OF, SF, ZF, AF, PF und CF
  - Ist eigentlich eine Sonderform von SUB. Er arbeitet intern genau wie SUB, schreibt aber das Ergebnis nicht in den ersten Operanden (fiktive Subtraktion)
- **PF** (Parity Flag) wird gesetzt, wenn bei der letzten Operation ein Bitmuster entstanden ist, das in den niederwertigen acht Bit aus einer geraden Anzahl von Einsen besteht (wird relativ selten benutzt, u.a. weil es nur acht Bit auswertet)
- **AF** (Auxiliary Flag) wird gesetzt, wenn bei der letzten Operation ein Übertrag von Bit 3 auf Bit 4, also ein Übertrag vom der unteren auf die obere Tetrade, entstanden ist
- **ZF** (Zero Flag) wird gesetzt, wenn das Ergebnis der letzten arithmetischen oder bitweise logischen Operation Null war
- **SF** (Sign Flag) wird gesetzt, wenn das Ergebnis der letzten Operation negativ war.
- **OF** (Overflow Flag) wird gesetzt, wenn bei der letzten Operation der vorzeichenbehaftete Wertebereich überschritten wird

# Wiederholung

- Bedingte Sprungbefehle
  - **Vorzeichenlose** Arithmetik: **below** und **above**
  - **Vorzeichenbehaftete** Arithmetik: **less** und **greater**

# SCASB, SCASW, SCASD

- **SCAS_**
  - CMP AL, <ES:EDI>  oder CMP AX, <ES:EDI> oder CMP EAX, <ES:EDI>
  - Vergleicht das Byte/Wort/Doppelwort in AL/AX/EAX mit dem an <ES:EDI> und setzt die Flags wie bei CMP
  - Falls **DF=0** dann wird **EDI mit Anzahl von gelesenen Bytes inkrementiert**
  - Falls **DF=1** dann wird **EDI mit Anzahl von gelesenen Bytes dekrementiert**

# CMPSB, CMPSW, CMPSD

- **CMPS_**
  - CMP <DS:ESI>, <ES:EDI>  (Byte/Wort/Doppelwort)
  - Vergleicht das Byte/Wort/Doppelwort an <DS:ESI> mit dem an <ES:EDI> und setzt die Flags wie bei CMP
  - Falls **DF=0** dann werden **ESI und EDI mit Anzahl von gelesenen Bytes inkrementiert**
  - Falls **DF=1** dann werden **ESI und EDI mit Anzahl von gelesenen Bytes dekrementiert**

# Bemerkung

- LODS, STOS, MOVS setzen keine Flags
- SCAS und CMPS setzen die Flags, da diese ein CMP ausführen

# Beispiel

```
MOV EDI, string + l - 1
MOV AL, 'e'
MOV ECX, l
STD
Cont_search:
      SCASB
      JE Found
LOOP Cont_search
;...
Found:
      INC EDI   ;gehe zurück zu dem Charakter bevor man EDI geändert hat
```

- Was is der Effekt?
  - Man findet die letzte Position im String *string* des Charakters `'e'`

# Wiederholungspräfixe für Stringbefehle

```
wiederholungsPräfix Stringbefehl
```

wobei `wiederholungsPräfix` folgende Werte haben kann:

- **REP** bewirkt, dass nach jeder Ausführung des nachstehenden Stringbefehls ECX dekrementiert und, falls ECX≠0, der Stringbefehl erneut ausgeführt wird.
- **REPE – Repeat While Equal |REPZ – Repeat While Zero**
  - bewirkt, dass nach jeder Ausführung des nachstehenden Stringbefehls ECX dekrementiert und, falls ECX≠0, der Stringbefehl erneut ausgeführt wird.
  - Hier existiert ein zweites Abbruchkriterium, nämlich die Ungleichheit der Operanden, d.h. Schleifen werden bei Ungleichheit der Operanden abgebrochen (wenn ZF=0)
- **REPNE (Repeat While Not Equal) | REPNZ (Repeat While Not Zero)**
  - bewirkt, dass nach jeder Ausführung des nachstehenden Stringbefehls ECX dekrementiert und, falls ECX≠0, der Stringbefehl erneut ausgeführt wird.
  - Hier existiert ein zweites Abbruchkriterium, nämlich die Gleichheit der Operanden, d.h. Schleifen werden bei Gleichheit der Operanden abgebrochen (wenn ZF=1)

# Bemerkung

- Man kann **REPE, REPZ,REPN,REPNZ** nur mit den Stringbefehlen **CMPS und SCAS** benutzen**.**
- Man kann zum Beispiel REP STOS_ benutzen um einen großen Speicherblock zu initialisieren (es ist schnell).

# Beispiel

```
Again:
     MOVSW
LOOP Again
```

ist äquivalent mit:

```
rep MOVSW
```

# Aufgaben

Löse folgende Aufgaben mit Stringbefehle:

1. Gegeben sei eine Bytefolge, die kleine Buchstaben enthält. Erstellen Sie eine neue Bytefolge, die die entsprechenden Großbuchstaben enthält.

# Aufgaben

2. Gegeben sei ein Doppelwort-String. Erstelle ein String, der in umgekehrter Reihenfolge folgende Bytes aus dem gegebenen String enthält:
   - die high Bytes aus der low Words, die zusätzlich Vielfache von 10 sind (in der vorzeichelose Arithmetik)

Beispiel:

- Gegeben: s dd 12345678h, 1A2B3C4Dh, 0FE98DC76h
- Man sollte den String d mit folgenden Werten erstellen: DCh, 3Ch
- Bem. 56h ist kein Vielfache von 10 und gehört deshalb nicht zu dem Ergebnis

# Aufgaben

3. Gegeben sei ein String mit Charakter. Bestimme die Anzahl der Vokale aus dem String.

# Hausaufgabe

1. Gegeben seien zwei String s1 und s2 mit derselben Länge und Datentyp. Erstelle ein String d durch Verschachteln von s1 und s2.

Bespiel: s1: 1,2,3,4,5

s2: 11,12,13,14,15

Dann ist d: 1, 11, 2, 12, 3, 13, 4, 14, 5, 15

2. Gegeben sei ein String von Quadwords. Berechne die Summe der strikt negativen low Bytes aus den high Doublewords.