**Curs 7 şi 8**

**Instrucţiuni ale limbajului de asamblare**

Contents

[1. Instrucţiuni de transfer al informaţiei 4](#_Toc26262618)

[1. Anweisungen zur Informationsübertragung 4](#_Toc26262619)

[1.1 Instrucţiuni de transfer de uz general 4](#_Toc26262620)

[1.1 TransferAnweisungen für allgemeine Zwecke 4](#_Toc26262621)

[1.2 Instrucţiunea de transfer al adreselor LEA 8](#_Toc26262622)

[1.2 Anweisung zur Übermittlung von LEA-Adressen 8](#_Toc26262623)

[1.3 Instrucţiuni asupra flagurilor 9](#_Toc26262624)

[1.3 Anweisungen auf Flaggen 9](#_Toc26262625)

[1.4 Instrucţiuni de conversie (distructivă) 10](#_Toc26262626)

[1.4 KonvertierungsAnweisungen (destruktiv) 10](#_Toc26262627)

[1.5 Impactul reprezentării little-endian asupra accesării datelor 12](#_Toc26262628)

[1.5 Auswirkung der Little-Endian-Darstellung auf den Datenzugriff 12](#_Toc26262629)

[2. Operaţii 14](#_Toc26262630)

[2. Operationen 14](#_Toc26262631)

[2.1 Operaţii aritmetice 14](#_Toc26262632)

[2.1 Rechenoperationen 14](#_Toc26262633)

[2.1.1 Add, adc, sub, subc 16](#_Toc26262634)

[2.1.1 Add, adc, sub, subc 16](#_Toc26262635)

[2.1.2 Mul, imul 16](#_Toc26262636)

[2.1.2 Mul, imul 16](#_Toc26262637)

[2.1.3 Div, idiv 17](#_Toc26262638)

[2.1.3 Div, idiv 17](#_Toc26262639)

[2.1.4 Neg, inc, dec 17](#_Toc26262640)

[2.1.4 Neg, inc, dec 17](#_Toc26262641)

[2.2 Operaţii logice pe biţi 17](#_Toc26262642)

[2.2 Logische Operationen an Bits 17](#_Toc26262643)

[2.3 Deplasări şi rotiri de biţi 18](#_Toc26262644)

[2.3 Bewegungen und Drehungen 18](#_Toc26262645)

[3. Ramificări, salturi, cicluri 18](#_Toc26262646)

[3. Äste, Sprünge, Zyklen 18](#_Toc26262647)

[3.1 Saltul necondiţionat 18](#_Toc26262648)

[3.1 Der bedingungslose Sprung 18](#_Toc26262649)

[3.1.1 Instrucţiunea JMP 19](#_Toc26262650)

[3.1.1 Die JMP-Anweisung 19](#_Toc26262651)

[3.1.2 Instrucţiuni de salt condiţionat şi necondiţionat 20](#_Toc26262652)

[3.1.2 Bedingte und unbedingte SprungAnweisungen 20](#_Toc26262653)

[3.1.3 Comparaţii între operanzi 22](#_Toc26262654)

[3.1.3 Vergleiche zwischen Operanden 22](#_Toc26262655)

[3.1.4 Instrucţiuni de ciclare 23](#_Toc26262656)

[3.1.4 SchleifAnweisungen 23](#_Toc26262657)

[3.2 Instrucţiunile CALL şi RET 24](#_Toc26262658)

[3.2 CALL- und RET-Anweisungen 24](#_Toc26262659)

[4. Instrucţiuni pe şiruri 26](#_Toc26262660)

[4. String Anweisungen 26](#_Toc26262661)

[4.1 Generalităţi privind instrucţiunile pe şiruri 26](#_Toc26262662)

[4.2 Instrucţiuni pe şiruri pentru transferul de date 28](#_Toc26262663)

[4.2 String-Anweisungen für die Datenübertragung 28](#_Toc26262664)

[4.3 Instrucţiuni pe șiruri pentru consultarea şi compararea datelor 31](#_Toc26262665)

[4.4 Execuţia repetată a unei instrucţiuni pe şiruri 33](#_Toc26262666)

[4.4 Wiederholte Ausführung eines String-Anweisung 33](#_Toc26262667)

[4.5 Utilizarea de operanzi pentru instrucţiuni pe şiruri 34](#_Toc26262668)

[4.5 Die Verwendung von Operanden für String-Anweisungen 34](#_Toc26262669)

|  |  |  |
| --- | --- | --- |
| Prezentăm forma generală a unui program în NASM, însoţită de un scurt exemplu: |  | Wir präsentieren die allgemeine Form eines Programms in NASM, begleitet von einem kurzen Beispiel: |
| **global** start ; solicităm asamblorului să confere vizibilitate globală simbolului denumit start. ; Eticheta start va fi punctul de intrare în program.  *; Wir bitten den Assembler, dem Symbol mit der Bezeichnung Start eine globale Sichtbarkeit zu verleihen. ; Das Start-Label ist der Einstiegspunkt in das Programm.*  **extern** ExitProcess, printf ; informăm asamblorul că simbolurile ExitProcess şi printf au ; provenienţă străină, evitând astfel semnalarea de erori cu privire ; la lipsa definirii acestora  *; Wir informieren die Assembler darüber, dass die ExitProcess- und printf-Symbole fremden*  *; Ursprungs sind, und vermeiden so die Meldung von Fehlern in Bezug auf fehlende Definition*  **import** ExitProcess kernel32.dll ; precizăm care sunt bibliotecile externe care definesc cele ; două simboluri: ExitProcess e parte a bibliotecii  ; kernel32.dll (bibliotecă standard a sistemului de operare)  *; Wir geben an, welche externen Bibliotheken die beiden Symbole definieren: ExitProcess ist*  *; Teil der kernel32.dll-Bibliothek (Standard-Betriebssystembibliothek).*  **import** printf msvcrt.dll; printf este funcţie standard C şi se regăseşte în biblioteca msvcrt.dll  ;(SO)  *; printf ist eine Standard-C-Funktion und befindet sich in der Bibliothek msvcrt.dll*  *;(Betriebssystem)*    **bits** 32 ; solicităm asamblarea pentru un procesor X86 (pe 32 biţi)  *; Wir benötigen die Assemblierung für einen X86-Prozessor (32 Bit)*  **segment code** use32 class = CODE; codul programului va fi emis ca parte a unui segment ; numit code  *; Der Programmcode wird als Teil eines Segments namens Code ausgegeben*  start:  ; apel printf (”Salut din ASM”) *; anruf printf ("Hallo von ASM")*  **push** dword string ; transmitem parametrul funcţiei printf (adresa şirului) pe ; stivă (aşa cere printf)  *; wir übertragen den Funktionsparameter printf (String-Adresse) auf den Stapel (so fragt*  *; printf)*  **call** [printf] ; printf este numele unei funcţii (etichetă = adresă, ; trebuie indirectată cu [])  *; printf ist der Name einer Funktion (Label = Adresse muss mit [] angesprochen werden)*  ; apel ExitProcess(0), 0 reprezentând „execuţie cu succes”  *; ExitProcess (0) aufrufen, 0 steht für „erfolgreiche Ausführung“*  **push** dword 0  **call** [ExitProcess]  **segment data** use32 class = DATA ; variabilele vor fi stocate în segmentul de date (denumit ; data)  *; Variablen werden im Datensegment gespeichert (Data genannt)*  string: db "Hallo von ASM!", 0 | | |
| 1. Instrucţiuni de transfer al informaţiei |  | 1. Anweisungen zur Informationsübertragung |
| 1.1 Instrucţiuni de transfer de uz general |  | 1.1 TransferAnweisungen für allgemeine Zwecke |
| **Tabelul 1**. *Instrucţiuni de transfer de uz general (*TransferAnweisungen für allgemeine Zwecke*)*   |  |  |  | | --- | --- | --- | | **MOV** *d,s* | <d> ← <s> (b-b, w-w, d-d) | - | | **PUSH** *s* | ESP = ESP - 4 şi depune <s> în stivă (s – dublucuvânt)  *und deponiere <s> im Stapel (s - Doppelwort)* | - | | **POP** *d* | extrage elementul curent din stivă şi îl depune în d (d – dublucuvânt) ESP = ESP + 4  *extrahiere das aktuelle Element aus dem Stapel und lege es in d (d - Doppelwort) ESP = ESP + 4* | - | | **XCHG** *d,s* | <d> **↔** <s> | - | | **[reg\_segment] XLAT** | AL ← < DS:[EBX+AL] > sau (*oder*) AL ← < reg\_segment:[EBX+AL] > | - | | **CMOVcc d, s** | <d> ← <s> dacă cc (cod condiţie) este adevărat  *wenn cc (bedingungscode) wahr ist* | - | | **PUSHA / PUSHAD** | Depune EDI, ESI, EBP, ESP, EBX, EDX, ECX şi EAX pe stivă  *Legt EDI, ESI, EBP, ESP, EBX, EDX, ECX und EAX auf dem Stapel ab* | - | | **POPA / POPAD** | Extrage EAX, ECX, EDX, EBX, ESP, EBP, ESI şi EDI de pe stivă  *Extrahiert EAX, ECX, EDX, EBX, ESP, EBP, ESI und EDI aus dem Stapel* | - | | **PUSHF** | Depune EFlags pe stivă  *Legt* EFlags *auf dem Stapel ab* | - | | **POPF** | Extrage vârful stivei şi îl depune în EFlags  *Extrahiert die Spitze des Stapels und legen Sie ihn in EFlags* | - | | **SETcc d** | <d> ← 1 dacă cc este adevărat, altfel <d> ← 0  *<d> ← 1, wenn cc wahr ist, andernfalls <d> ← 0* | - | | | |
| Dacă operandul destinaţie al instrucţiunii **MOV** este unul dintre cele 6 registre de segment, atunci sursa trebuie să fie unul dintre cele opt registre generale de 16 biţi ale UE sau o variabilă de memorie. Încărcătorul de programe al sistemului de operare pre-iniţializează în mod automat registrele de segment, iar schimbarea valorilor acestora, deşi posibilă din punct de vedere al procesorului, nu aduce nici o utilitate (un program este limitat la a încărca doar valori de selectori care indică înspre segmente preconfigurate de către sistemul de operare, fără a putea să definească segmente adiţionale).  Instrucţiunile **PUSH** şi **POP** au sintaxa:  **PUSH** *s*  şi respectiv **POP** *d*  Operanzii trebuie să fie reprezentaţi pe dublucuvânt, deoarece stiva este organizată pe dublucuvinte. Stiva creşte de la adrese mari spre adrese mici, din 4 în 4 octeţi, ESP punctând întotdeauna spre dublucuvântul din vârful stivei.  Funcţionarea acestor instrucţiuni poate fi ilustrată prin intermediul unei secvenţe echivalente de instrucțiuni **MOV** şi **ADD** sau **SUB**: |  | Wenn der Zieloperand der **MOV**-Anweisung eines der 6 Segmentregister ist, muss die Quelle eines der acht 16-Bit-Allgemeinregister des UE oder eine Speichervariable sein. Das Programmladeprogramm des Betriebssystems initialisiert die Segmentregister automatisch vor, und das Ändern ihrer Werte bringt, obwohl dies aus Sicht des Prozessors möglich ist, kein Hilfsprogramm (ein Programm ist darauf beschränkt, nur ausgewählte Werte zu laden, die dies anzeigen) auf vom Betriebssystem vorkonfigurierte Segmente, ohne zusätzliche Segmente definieren zu können).  Die Anweisungen **PUSH** und **POP** haben folgende Syntax:  **PUSH** *s*und **POP** *d*  Die Operanden müssen auf Doppelwort dargestellt werden, da der Stapel auf dem Doppelwörte organisiert ist. Der Stapel wächst von großen Adressen zu kleinen Adressen von 4 bis 4 Bytes, wobei ESP immer auf das Doppelwort oben im Stapel zeigt. Die Funktionsweise diesen Anweisungen kann anhand einer äquivalenten Folge von Anweisungen **MOV** und **ADD** oder **SUB** veranschaulicht werden: |
| |  |  |  | | --- | --- | --- | | **push** EAX ⬄ | **sub** ESP, 4  **mov** [ESP], EAX | ; pregătim (alocăm) spaţiu pentru a stoca valoarea  *; Wir bereiten Speicherplatz vor, um den Wert zu speichern* ; stocăm valoarea în locaţia alocată  *; Wir speichern den Wert an dem zugewiesenen Ort* | | **pop** EAX ⬄ | **mov** EAX, [ESP]  **add** ESP, 4 | ; încărcăm în EAX valoarea din vârful stivei  *; Wir laden in EAX den Wert von der Spitze des Stapels* ; eliberăm locaţia  *; Wir räumen den Ort* | | | |
| Aceste instrucţiuni permit doar depunerea şi extragerea de valori reprezentate pe cuvânt şi dublucuvânt. Ca atare,  **PUSH** AL  nu reprezintă o instrucţiune validă (*syntax error*), deoarece operandul nu are voie să aibă o valoare pe octet. Pe de altă parte, secvenţa de instrucţiuni: |  | Diese Anweisungen erlauben nur das Ablegen und Extrahieren von Werten, die durch word und double dargestellt werden. Als solches  **PUSH** AL  ist kein gültiger Anweisung (*syntax error*), da der Operand keinen Byte-Wert haben darf. Zum anderen die Reihenfolge der Anweisungen: |
| **PUSH** AX ; depunem AX (*wir hinterlegen AX*)  **PUSH** EBX ; depunem EBX (*wir hinterlegen EBX*)  **POP** ECX ; ECX ← dublucuvântul din vârful stivei (valoarea lui EBX)  *; ECX ← das Doppelwort oben im Stapel (der Wert von EBX)*  **POP** DX ; DX ← cuvântul rămas în stivă (deci valoarea lui AX)  *; DX ← das Wort im Stapel (also der Wert von AX)* | | |
| este corectă şi echivalentă prin efect cu |  | ist korrekt und äquivalent zu |
| **MOV** ECX, EBX  **MOV** DX, AX | | |
| Adiţional acestei constrângeri (inerentă tuturor procesoarelor x86), sistemul de operare impune ca operarea stivei să fie obligatoriu făcută doar prin accese pe dublucuvânt sau multipli de dublucuvânt, din motive de compatibilitate între programele de utilizator şi nucleul şi bibliotecile de sistem. Implicaţia acestei constrângeri este că o instrucţiune de forma **PUSH** operand16 sau **POP** operand16 (de exemplu **PUSH** word 10), deşi este suportată de către procesor şi asamblată cu succes de către asamblor, nu este recomandată, putând cauza ceea ce poartă numele de eroare de dezaliniere a stivei: stiva este corect aliniată dacă şi numai dacă valoarea din registrul ESP este în permanenţă divizibilă cu 4!  Instrucţiunea **XCHG** permite interschimbarea conţinutului a doi operanzi de aceeaşi dimensiune (octet, cuvânt sau dublucuvânt), cel puţin unul dintre ei trebuind să fie registru. Sintaxa ei este: |  | Zusätzlich zu dieser Einschränkung (die allen x86-Prozessoren inhärent ist) erfordert das Betriebssystem, dass die Stapeloperation aus Gründen der Kompatibilität zwischen Benutzerprogrammen und dem Kernel und den Systembibliotheken nur durch Zugriffe auf Doppeltwörte oder Doppeltwörte Vielfache obligatorisch ist. Die Implikation dieser Einschränkung ist, dass ein Anweisung der Form **PUSH** Operand16 oder **POP** Operand16 (zum Beispiel **PUSH** Wort 10), obwohl er vom Prozessor unterstützt und vom Assembler erfolgreich zusammengestellt wird, nicht empfohlen wird, was zu einem sogenannten Fehlausrichtungsfehler führen kann des Stapels: Der Stapel ist genau dann richtig ausgerichtet, wenn der Wert im ESP-Register dauerhaft durch 4 teilbar ist! Die **XCHG** Anweisung ermöglicht den Austausch von Inhalten zweier Operanden derselben Größe (Byte, Wort oder Doppeltwort), von denen mindestens einer eine Registrierung sein muss. Ihre Syntax lautet: |
| **XCHG** *operand1, operand2* | | |
| Instrucțiunea **XLAT** „traduce” octetul din AL într-un alt octet, utilizând în acest scop o tabelă de corespondență creată de utilizator, numită *tabelă de translatare.* Instrucțiunea are sintaxa  **[reg\_segment] XLAT**  *tabelă\_de\_translatare* este adresa directă a unui şir de octeţi. Instrucţiunea **XLAT** pretinde la intrare adresa far a tabelei de translatare furnizată sub unul din următoarele două moduri:  - DS:EBX (implicit, dacă lipseşte precizarea registrului segment)  - registru\_segment:EBX, dacă registrul segment este precizat explicit.  Efectul instrucţiunii **XLAT** este înlocuirea octetului din AL cu octetul din tabelă ce are numărul de ordine valoarea din AL (primul octet din tabelă are indexul 0). EXEMPLU: pag.111-112 (curs).  De exemplu, secvenţa: |  | Die **XLAT**-Anweisung „übersetzt“ das Byte von AL in ein anderes Byte unter Verwendung einer vom Benutzer erstellten Mailing-Tabelle, die zu diesem Zweck als *Übersetzungstabelle* bezeichnet wird. Die Anweisung hat die Syntax  **[reg\_segment] XLAT**  *Übersetzungstabelle* ist die direkte Adresse einer Folge von Bytes. Die **XLAT**- Anweisung fordert am Eingang die FAR Adresse der Übersetzungstabelle an, die in einem der beiden folgenden Modi bereitgestellt wird:   * DS:EBX (Standard, wenn die Segmentspezifikation fehlt); * Segment\_Register:EBX, wenn das Segmentregister explizit angegeben ist.   Die **XLAT**-Anweisung bewirkt, dass das Byte in der AL durch das Byte in der Tabelle ersetzt wird, dessen Ordnungsnummer der Wert in der AL ist (das erste Byte in der Tabelle hat den Index 0). BEISPIEL: Seite 111-112 (Kurs).  Zum Beispiel die Reihenfolge: |
| **mov** EBX, Tabela  **mov** AL, 6  ES **xlat ;** AL ← < ES:[EBX+6] > | | |
| depune conţinutul celei de-a 7-a locaţii de memorie (de index 6) din *Tabela* în AL.  Dăm un exemplu de secvenţă care translatează o valoare zecimală 'număr' cuprinsă între 0 şi 15 în cifra hexazecimală (codul ei ASCII) corespunzătoare: |  | speichert den Inhalt des 7. Speicherplatzes (Index 6) von *Tabelle* in AL.  Wir geben ein Beispiel für eine Sequenz, die eine Dezimalwert 'număr' zwischen 0 und 15 in die entsprechende Hexadezimalzahl (ihren ASCII-Code) übersetzt: |
| **segment data** use32  . . .  TabHexa db '0123456789ABCDEF'  . . .  **segment code** use32  **mov** EBX, TabHexa  . . .  **mov** AL, număr  **xlat** ; AL ← < DS:[EBX+AL] > | | |
| O astfel de strategie este des utilizată şi se dovedeşte foarte utilă în cadrul pregătirii pentru tipărire a unei valori numerice întregi (practic este vorba despre o conversie din *valoare numerică registru* în *string de tipărit*). |  | Eine solche Strategie wird oft verwendet und erweist sich als sehr nützlich bei der Vorbereitung zum Drucken eines ganzen numerischen Werts (im Grunde handelt es sich um eine Umwandlung eines *numerischen Wertregisters* in eine *Druckzeichenfolge*). |
| 1.2 Instrucţiunea de transfer al adreselor LEA |  | 1.2 Anweisung zur Übermittlung von Adressen: LEA |
| |  |  |  | | --- | --- | --- | | **LEA** *reg\_general, mem* | *reg\_general* ← *offset*(*mem*) | - | | | |
| Instrucţiunea **LEA** (***L****oad* ***E****ffective* ***A****ddress*) transferă deplasamentul operandului din memorie *mem* în registrul destinaţie. De exemplu:  **lea** EAX, [v]  încarcă în EAX *offset*-ul variabilei v, instrucţiune echivalentă cu:    **mov** EAX, v  Instrucţiunea **LEA** are însă avantajul că operandul sursă poate fi o expresie de adresare (spre deosebire de instrucțiunea **mov** care nu acceptă pe post de operand sursă decât o variabilă cu adresare directă într-un astfel de caz). De exemplu, instrucţiunea: |  | Die **LEA**-Anweisung (***L****oad* ***E****ffective* ***A****ddress*) überträgt das Offset des Operanden aus dem *Mem*-Speicher in das Zielregister. Zum Beispiel:  **lea** EAX, [v]  ladet das *Offset* von die Variablen v in EAX. Dieser Anweisung entspricht:  **mov** EAX, v  Der **LEA**-Anweisung hat jedoch den Vorteil, dass der Quelloperand ein Adressausdruck sein kann (im Gegensatz zum der **mov** Anweisung, der den Quelloperanden in einem solchen Fall nicht als direkte Adressvariable akzeptiert). Zum Beispiel die Anweisung: |
| **lea** EAX, [EBX + v - 6] | | |
| nu are ca echivalent direct o singură instrucţiune **MOV**, instrucţiunea |  | hat nicht als direktes Äquivalent eine einzelne **MOV**-Anweisung, die Anweisung |
| **mov** EAX, EBX + v – 6 | | |
| fiind incorectă sintactic deoarece expresia EBX + v - 6 nu este determinabilă la momentul asamblării.  **LEA** permite utilizarea directă a valorilor deplasamentelor care rezultă în urma calculelor de adrese (în contrast cu folosirea memoriei indicate de către acestea). Astfel, **LEA** se evidenţiază prin versatilitate şi eficienţă sporite. Instrucţiunea **LEA** este considerată versatilă datorită faptului că permite combinarea unei înmulţiri cu adunări de valori stocate în regiştri şi/sau cu valori constante. Pe de altă parte, instrucţiunea **LEA** este considerată eficientă datorită faptului că întregul calcul se execută într-o singură instrucţiune, fără a ocupa circuitele ALU care rămân astfel disponibile pentru alte operaţii (timp în care calculul de adresă este efectuat de către circuite specializate, separate, care fac parte din BIU).  Exemplu: înmulţirea unui număr cu 10: |  | ist syntaktisch falsch, da der Ausdruck EBX + v - 6 zum Zeitpunkt der Assemblierung nicht bestimmbar ist.  **LEA** ermöglicht die direkte Verwendung der aus den Adressberechnungen resultierenden Offsetswerte (im Gegensatz zur Verwendung des von ihnen angegebenen Speichers). Somit zeichnet sich der **LEA** durch seine gesteigerte Vielseitigkeit und Effizienz aus. Die **LEA**-Anweisung wird als vielseitig angesehen, da es die Kombination einer Multiplikation mit den in den Registern gespeicherten Wertegruppen und / oder mit konstanten Werten ermöglicht. Andererseits wird der **LEA**-Anweisung als effizient angesehen, da die gesamte Berechnung in einem einzigen Anweisung ausgeführt wird, ohne die ALU-Schaltungen zu belegen, die somit für andere Operationen verfügbar bleiben (während derer die Adressberechnung von getrennten spezialisierten Schaltungen ausgeführt wird, die Teil der BIU sind).  Beispiel: Multiplikation einer Zahl mit 10: |
| **mov** EAX, [număr] ; EAX ← valoarea variabilei număr (*die Wert der Variable* număr)  **lea** EAX, [EAX \* 2] ; EAX ← număr \* 2  **lea** EAX, [EAX \* 4 + EAX]; EAX ← (EAX \* 4) + EAX = EAX \* 5 = (număr \* 2) \* 5 | | |
| 1.3 Instrucţiuni asupra flagurilor |  | 1.3 Anweisungen auf Flaggen |
| Următoarele patru instrucţiuni sunt *instrucţiuni de transfer* al indicatorilor:  Instrucţiunea **LAHF** (***L****oad register* ***AH*** *from* ***F****lags*) copiază indicatorii SF (*Sign Flag*), ZF (*Zero Flag*), AF (*Adjust Flag*), PF (*Parity Flag*) şi CF (*Carry Flag*) din registrul de *flag*-uri în biţii 7, 6, 4, 2 şi respectiv 0 ai registrului AH. Conţinutul biţilor 5, 3 şi 1 este nedefinit. Indicatorii nu sunt afectaţi în urma acestei operaţii de transfer (în sensul că instrucţiunea LAHF nu este ea însăşi generatoare de efecte asupra unor flag-uri – ea doar copiază valorile flag-urilor).  Instrucţiunea **SAHF** (***S****tore register* ***AH*** *into* ***F****lags*) transferă biţii 7, 6, 4, 2 şi 0 ai registrului AH în indicatorii SF, ZF, AF, PF şi respectiv CF, înlocuind valorile anterioare ale acestor indicatori.  Instrucţiunea **PUSHF** transferă toţi indicatorii în vârful stivei (conţinutul registrului Flags se transferă în vârful stivei). Indicatorii nu sunt afectaţi în urma acestei operaţii. Instrucţiunea **POPF** extrage cuvântul din vârful stivei şi transferă din acesta indicatorii corespunzători în registrul de flag-uri.  Limbajul de asamblare pune la dispoziţia programatorului nişte *instrucţiuni de setare* a valorii indicatorilor de condiţie, pentru ca programatorul să poată influenţa după dorinţă modul de acţiune al instrucţiunilor care exploatează *flag*-uri. |  | Die folgenden vier Anweisungen sind *Anweisungen zum Übertragen* der Indikatoren: Die Anweisung **LAHF** (***L****ade* ***R****egister* ***AH*** *von* ***F****lags*) kopiert die Flags SF (*Sign Flag*, Vorzeichenflagge), ZF (*Zero Flag*, Nullflagge), AF (*Adjust Flag*, Einstellflagge), PF (Paritätsflagge) und CF (Übertragflagge) aus dem 7-Bit-Flagregister. 6, 4, 2 bzw. 0 des AH-Registers. Der Inhalt der Bits 5, 3 und 1 ist unbestimmt. Die Indikatoren sind von dieser Übertragungsoperation nicht betroffen (was bedeutet, dass die LAHF-Anweisung selbst kein Flag-verursachender Effekt ist - sie kopiert nur die Werte der Flags). Die Anweisung **SAHF** (Register AH in Flags speichern) überträgt die Bits 7, 6, 4, 2 und 0 des AH-Registers an die Indikatoren SF, ZF, AF, PF und CF und ersetzt die vorherigen Werte dieser Indikatoren.  Die **PUSHF**-Anweisung überträgt alle Indikatoren an den Anfang des Stapels (der Inhalt des Flags-Registers wird an den Anfang des Stapels übertragen). Die Anzeigen sind von diesem Vorgang nicht betroffen. Die **POPF**-Anweisung extrahiert das Wort oben im Stapel und überträgt die entsprechenden Flags von dort in das Flagregister. Die Assemblersprache liefert dem Programmierer Anweisungen zum Einstellen des Werts der Bedingungsindikatoren, so dass der Programmierer den Aktionsmodus der Anweisungen beeinflussen kann, die Flags wie gewünscht bedienen. |
| |  |  |  | | --- | --- | --- | | **CLC** | CF = 0 | CF (*Carry Flag*) | | **CMC** | CF = ~CF | CF (*Carry Flag*) | | **STC** | CF = 1 | CF (*Carry Flag*) | | **CLD** | DF = 0 | DF (*Direction Flag*) | | **STD** | DF = 1 | DF (*Direction Flag*) | | | |
| 1.4 Instrucţiuni de conversie (distructivă) |  | 1.4 Konvertierungsanweisungen (destruktiv) |
| |  |  |  | | --- | --- | --- | | **CBW** | conversie octet conţinut în AL la cuvânt în AX (extensie de semn)  *Byte-Inhalt von AL nach Wort nach AX konvertieren (Vorzeichenerweiterung)* | - | | **CWD** | conversie cuvânt conţinut în AX la dublu cuvânt în DX:AX (extensie de semn)  *Konvertieren von Wortinhalten in AX in Doppelwort in DX:AX (Vorzeichenerweiterung)* | - | | **CWDE** | conversie cuvânt din AX în dublucuvânt în EAX (extensie de semn)  *Konvertieren dem Wort von AX in Doppelwort in EAX (Zeichenerweiterung)* | - | | **MOVZX d, s** | încarcă în *d* (REGISTRU !), de dimensiune mai mare decât *s*, conținutul lui *s* fără semn  *Ladet in* d *(REGISTER!), größer als* s*, den Inhalt von* s *ohne Vorzeichen* | - | | **MOVSX d, s** | încarcă în *d* (REGISTRU !), de dimensiune mai mare decât *s*, conținutul lui *s* cu semn  *Ladet in* d *(REGISTER!), größer als* s*, den Inhalt von* s *mit Vorzeichen* | - | | | |
| Instrucţiunea **CBW** converteşte octetul cu semn din AL în cuvântul cu semn AX (extinde bitul de semn al octetului din AL la nivelul cuvântului din AX, modificând distructiv conţinutul registrului AH). De exemplu: |  | Die **CBW**-Anweisung konvertiert das Vorzeichenbyte von AL in das Wort mit dem Vorzeichen AX (erweitert das Vorzeichenbit des Bytes von AL auf das Wort von AX, wodurch der Inhalt des Registers AH destruktiv geändert wird). Zum Beispiel: |
| **mov** al, -1 ; AL = 0FFh  **cbw** ;extinde valoarea (octet) -1 din AL în valoarea (cuvânt) -1 ;din AX (0FFFFh).  *;erweitert den Wert (Byte) -1 von AL auf den Wert (Wort) ;-1 von AX (0FFFFh).* | | |
| Analog, pentru a realiza conversia cu semn de la cuvânt la dublucuvânt, instrucţiunea **CWD** extinde cuvântul cu semn din AX în dublucuvântul cu semn DX:AX. Exemplu: |  | In ähnlicher Weise, um die Wort-zu-Doppelwort-Umwandlung durchzuführen, erweitert die **CWD**-Anweisung von den Wortzeichen von AX zu dem Doppelwortzeichen DX:AX. Beispiel: |
| **mov** ax,-10000; AX = 0D8F0h  **cwd** ; obţine valoarea (*Holen den Wert*) -10000 in ; DX:AX (DX = 0FFFFh ; AX = 0D8F0h)  **cwde** ; obţine valoarea (*Holen den Wert*) -10000 in EAX ; (EAX = 0FFFFD8F0h) | | |
| Conversia **fără semn** se realizează prin zero-izarea octetului sau cuvântului superior al valorii de la care s-a plecat. (de exemplu, prin **mov** AH,0 sau **mov** DX,0 – efect similar se obţine prin aplicarea instrucţiunii **MOVZX**).  De ce coexistă **CWD** cu **CWDE** ? **CWD** trebuie să rămână din raţiuni de *backwards compatibility* şi din raţiuni de funcţionalitate a instrucţiunilor **(I)MUL** şi **(I)DIV**. |  | Die Konvertierung ohne Vorzeichen erfolgt durch Nullsetzen des Bytes oder des oberen Wortes des Werts, von dem aus es gestartet wurde. (Zum Beispiel bei **mov** AH, 0 oder **mov** DX, 0 - Ein ähnlicher Effekt wird durch Anwenden des **MOVZX**-Anweisung).  Warum koexistiert **CWD** mit **CWDE**? Die **CWD** muss aus Gründen der Abwärtskompatibilität (*backwards compatibility]*) und aus Gründen der Funktionalität der Anweisungen (**I**)**MUL** und (**I**)**DIV** erhalten bleiben. |
| MOV AH, 0C8h  MOVSX EBX, AH ; EBX = FFFFFFC8h  MOVSX AX, [v] ; MOVSX AX, byte ptr DS:[offset v]  MOVZX EDX, AH ; EDX = 000000C8h  MOVZX EAX, [v] ; *syntax error – op.size not specified* | | |
| Atenţie ! NU sunt acceptate sintactic: |  | Achtung! NICHT syntaktisch akzeptiert: |
| |  |  |  | | --- | --- | --- | | CBD | CWDE EBX, BX | MOVSX EAX, [v] | | CWB | CWD EDX,AX | MOVZX EAX, [EBX] | | CDW | MOVZX AX, BX | MOVSX dword [EBX], AH | | CDB | MOVSX EAX, -1 | CBW BL | | | |
| 1.5 Impactul reprezentării little-endian asupra accesării datelor |  | 1.5 Auswirkung der Little-Endian-Darstellung auf den Datenzugriff |
| Dacă programatorul utilizează datele consistent cu dimensiunea de reprezentare stabilită la definire (exemplu: accesarea octeţilor drept octeţi şi nu drept secvenţe de octeţi interpretate ca şi cuvinte sau dublucuvinte, accesarea de cuvinte ca şi cuvinte şi nu ca perechi de octeţi, accesarea de dublucuvinte ca şi dublucuvinte şi nu ca secvenţe de octeţi sau de cuvinte) atunci instrucţiunile limbajului de asamblare vor ţine cont în mod AUTOMAT de modalitatea de reprezentare little-endian. Ca urmare, dacă se respectă această condiţie programatorul nu trebuie să intervină suplimentar în nici un fel pentru a asigura corectitudinea accesării şi manipulării datelor utilizate. Exemplu: |  | Wenn der Programmierer die Daten verwendet, die mit der in der Definition festgelegten Repräsentationsdimension übereinstimmen (zum Beispiel: Zugreifen auf Bytes als Bytes und nicht als Bytefolgen, die als Wörter oder Doppelwörter interpretiert werden, Zugreifen auf Wörter als Wörter und nicht als Paare von Bytes, Zugreifen auf Doppelwörter als und Doppelwörter und nicht als Byte- oder Wortfolge), dann werden die Anweisungen der Assemblersprache AUTOMATISCH den Little-Endian-Darstellungsmodus berücksichtigen. Wenn diese Bedingung erfüllt ist, darf der Programmierer in keiner Weise eingreifen, um den korrekten Zugriff und die Manipulation der verwendeten Daten sicherzustellen. Beispiel: |
| a db ‘d’, -25, 120  b dw -15642, 2ba5h  c dd 12345678h  …  **mov** AL, [a] ;se încarcă în AL codul ASCII al caracterului ‘d’  *; Man ladet den ASCII-Code des Zeichens 'd' in AL*  **mov** BX, [b] ;se încarcă în BX valoarea -15642; ordinea octeţilor în BX va fi însă inversată faţă de reprezentarea în memorie, deoarece numai reprezentarea *în memorie* foloseşte reprezentarea *little-endian*! În regiştri datele sunt memorate conform reprezentării structurale normale, echivalente unei reprezentări *big endian*.  *; Man ladet in BX den Wert -15642; Die Reihenfolge der Bytes in BX wird jedoch von der Darstellung im Speicher umgekehrt, da nur die Darstellung im Speicher die* Little-Endian*-Darstellung verwendet! In den Registern werden die Daten entsprechend der normalen Strukturdarstellung gespeichert, die einer* Big-Endian*-Darstellung entspricht.*  **mov** EDX, [c] ;se încarcă în EDX valoarea dublucuvânt 12345678h  *; der Doppelwort 12345678h wird in EDX geladen* | | |
| Dacă însă se doreşte accesarea sau interpretarea datelor sub o formă diferită faţă de modalitatea de definire atunci trebuie utilizate conversii explicite de tip. În momentul utilizării conversiilor explicite de tip programatorul trebuie să îşi asume însă întreaga responsabilitate a interpretării şi accesării corecte a datelor. În astfel de situaţii programatorul este obligat să conştientizeze particularităţile de reprezentare little-endian (ordinea de plasare a octeţilor în memorie) şi să utilizeze modalităţi de accesare a datelor în conformitate cu aceasta (cartea de curs, pag. 120-122). |  | Wenn jedoch auf die Daten in einer anderen als der definierten Form zugegriffen oder diese interpretiert werden sollen, müssen explizite Typkonvertierungen verwendet werden. Bei der Verwendung expliziter Typkonvertierungen muss der Programmierer jedoch die volle Verantwortung für die korrekte Interpretation und den korrekten Zugriff auf die Daten übernehmen. In solchen Situationen ist der Programmierer verpflichtet, die Besonderheiten der Little-Endian-Darstellung (die Reihenfolge der Platzierung der Bytes im Speicher) zu kennen und Möglichkeiten des Zugriffs auf die Daten entsprechend zu verwenden (Kursbuch, Seiten 120-122). |
| **segment** data  **a dw 1234h** ; datorită reprezentării little-endian, în memorie octeţii sunt plasaţi astfel:  *;Dank der* Little-Endian*-Darstellung werden die Bytes wie folgt gespeichert:*  **b dd 11223344h** ; 34h 12h 44h 33h 22h 11h  ; *adresa* a a + 1 b b + 1 b + 2 b + 3  **c db -1**  **segment** code  **mov** AL, byte [a+1] ; accesarea lui a drept octet, efectuarea calculului de adresă a+1, selectarea octetului de la adresa a+1 (octetul de valoare 12h) şi transferul său în registrul AL.  *; Zugriff als Byte, Ausführen der Adressberechnung von + 1, Auswählen des Bytes aus der Adresse von + 1 (Wertbyte 12h) und Übertragen des Bytes in das AL-Register.*  **mov** DX, word [b+2] ; DX:=1122h  **mov** dx, word **mov** DX, word [a+4] ; DX:=1122h deoarece b+2 = a+4 , în sensul că aceste expresii de tip pointer desemnează aceeaşi adresă şi anume adresa octetului 22h.  *; DX: = 1122 ; DX:=1122h, weil b+2 = a+4, in dem Sinne, dass diese Zeigerausdrücke dieselbe Adresse bezeichnen, nämlich die 22h-Byteadresse.*  **mov** DX, [a+4] ; această instrucţiune este echivalentă cu cea de mai sus, nefiind realmente  ; necesară utilizarea operatorului de conversie WORD  *;Diese Anweisung ist äquivalent zu der obigen, nicht wirklich notwendigen Anweisung, die den WORD-Konvertierungsoperator verwendet*  **mov** BX, [b] ; BX:=3344h  **mov** BX, [a+2] ; BX:=3344h, deoarece ca adrese b = a+2.  *; BX:=3344h, weil als Adressen b = a+2.*  **mov** ECX, dword [a] ; ECX:=33441234h, deoarece dublucuvântul ce începe la adresa a este format din octeţii 34h 12h 44h 33h care (datorită reprezentării little-endian) înseamnă de fapt dublucuvântul 33441234h.    *; ECX: = 33441234h, weil der Doppelwort bei Adresse a aus den*  *Vorkommen von 34h 12h 44h 33h besteht, was (aufgrund der Little- Endian-Darstellung) tatsächlich bedeutet: das Doppelwort 33441234h.*  **mov** EBX, [b] ; EBX := 11223344h  **mov** AX, word [a+1] ; AX := 4412h  **mov** EAX, word [a+1] ; AX := 22334412h  **mov** DX, [c-2] ; DX := 1122h deoarece (*weil*) c-2 = b+2 = a+4  **mov** BH, [b] ; BH := 44h  **mov** CH, [b-1] ; CH := 12h  **mov** CX, [b+3] ; CX := 0FF11h | | |
| 2. Operaţii |  | 2. Operationen |
| 2.1 Operaţii aritmetice |  | 2.1 Rechnenoperationen |
| Operanzii sunt reprezentaţi în Complementul lui 2. Când realizează adunările şi scăderile, microprocesorul „vede” doar configuraţii de biţi şi nu numere cu semn sau fără. Regulile de efectuare a adunării şi scăderii presupun adunarea de configuraţii binare, fără a fi nevoie de a interpreta operanzii drept cu semn sau fără semn anterior efectuării operaţiei! Deci, la nivelul acestor instrucţiuni, interpretarea „cu semn” sau „fără semn” rămâne la latitudinea programatorului, nefiind nevoie de instrucţiuni separate pentru adunarea/scăderea cu semn faţă de adunarea/scăderea fără semn.  Adunarea şi scăderea se efectuează întotdeauna la fel (adunând sau scăzând configuraţii binare) indiferent de semnul (interpretarea) acestor configuraţii! După cum vom vedea acest lucru nu este valabil şi pentru înmulţire şi împărţire. În cazul acestor operaţii trebuie să ştim apriori dacă operanzii vor fi interpretaţi drept cu semn sau fără semn. De exemplu, fie doi operanzi A şi B reprezentaţi fiecare pe câte un octet: |  | Die Operanden sind im Zweierkomplement dargestellt. Beim Addieren und Subtrahieren „sieht“ der Mikroprozessor nur Bitkonfigurationen und keine Zahlen mit oder ohne Vorzeichen. Die Addition- und Subtraktionsregeln erfordern das Hinzufügen von Binärkonfigurationen, ohne dass die Operanden als Vorzeichen oder ohne Vorzeichen interpretiert werden müssen, bevor die Operation ausgeführt wird! Auf der Ebene dieser Anweisungen liegt die Interpretation „mit Vorzeichen“ oder „ohne Vorzeichen“ im Ermessen des Programmierers und erfordert keine separaten Anweisungen für die Addition / Subtraktion mit dem Vorzeichen gegenüber der Addition / Subtraktion ohne das Vorzeichen. Das Addition und Subtraktion erfolgt immer gleich (durch Addition oder Subtraktion von binären Konfigurationen), unabhängig vom Vorzeichen (Interpretation) dieser Konfigurationen! Wie wir sehen werden, gilt dies nicht für Multiplikation und Division. Bei diesen Operationen müssen wir a priori wissen, ob die Operanden als Vorzeichen oder ohne Vorzeichen interpretiert werden. Beispielsweise sind in jedem Byte zwei Operanden A und B dargestellt: |
| A = 9Ch = 10011100b (= 156 în interpretarea fără semn şi -100 în interpretarea cu semn)  *(= 156 in der Interpretation ohne Vorzeichen und -100 in der Interpretation mit Vorzeichen)*  B = 4Ah = 01001010b (= 74 atât în interpretarea fără semn cât şi în interpretarea cu semn)  *(= 74 sowohl in der vorzeichenlosen als auch in der vorzeichendeutung)* | | |
| Microprocesorul realizează adunarea C = A + B şi obţine: |  | Der Mikroprozessor realisiert die Addition C = A + B und erhält: |
| C = E6h = 11100110b (= 230 în interpretarea fără semn şi -26 în interpretarea cu semn)  *(= 230 in der Interpretation ohne Vorzeichen und -26 in der Interpretation mit Vorzeichen)* | | |
| Se observă deci că simpla adunare a cuvintelor binare (fără a ne fixa neapărat asupra unei interpretări anume la momentul efectuării adunării) asigură corectitudinea rezultatului obţinut, atât în interpretarea cu semn cât şi în cea fără semn. |  | Es wird somit festgestellt, dass die einfache Addition der Binärwörter (ohne dass zum Zeitpunkt des Zusammenbaus unbedingt eine bestimmte Interpretation im Vordergrund steht) die Richtigkeit des erhaltenen Ergebnisses sowohl in der vorzeichenbehafteten als auch in der nicht vorzeichenbehafteten Interpretation gewährleistet. |
| 2.1.1 Add, adc, sub, subc |  | 2.1.1 Add, adc, sub, subc |
| **add** *dest*, *src* ; *dest* := *dest* + *src*  **adc** *dest*, *src* ; *dest* := *dest* + *src* + Carry  **sub** *dest*, *src* ; *dest* := *dest* - *src*  **sbb** *dest*, *src* ; *dest* := *dest* - *src* - Carry | | |
| * *dest* - reg8/16/32, mem8/16/32 * src – reg8/16/32, mem8/16/32, *dată\_imediată* * operaţiile aritmetice şi logice afectează următorii indicatori de condiţie: CF, AC, ZF, SF, OF, PF * eventuala depăşire a capacităţii se verifică de către programator (atenţie la forma de reprezentare cu / fără semn): CF – depăşire la operaţiile fără semn; OF – depăşire la operaţiile cu semn. |  | * *dest* - reg8/16/32, mem8/16/32 * src – reg8/16/32, mem8/16/32, *Sofort\_Data* * Arithmetische und logische Operationen wirken sich auf die folgenden Bedingungsindikatoren (Flagen) aus: CF, AC, ZF, SF, OF, PF * die eventuelle Überkapazität wird vom Programmierer überprüft (Beachtung der Darstellungsform mit / ohne Vorzeichen): CF - Überschreitung der Operationen ohne Vorzeichen; OF - Überschreitung der Vorzeichenoperationen. |
| 2.1.2 Mul, imul |  | 2.1.2 Mul, imul |
| Instrucţiunea **mul** realizează înmulțirea numerelor întregi fără semn, iar instrucţiunea **imul** realizează înmulțirea numerelor întregi cu semn. |  | Der **mul**-Anweisung führt die Multiplikation von ganzen Zahlen ohne Vorzeichen aus, und der Anweisung **imul** führt die Multiplikation von ganzen Zahlen mit einem Vorzeichen aus. |
| **mul** *src* ; *acc* := *acc*L0 × *src*  **imul** *src* ; *acc* := *acc*L0 × *src*  **imul** *dest*, *src1*, *imm*\_*src* ; *dest* := *src1* × *imm*\_*src*  **imul** *dest*, *imm*\_*src* ; *dest* := *dest* × *imm*\_*src*  **imul** *dest*, *src* ; *dest* := *dest* × *src*  • *src* – reg8/16/32, mem8/16/32  • *acc* – AX, DX:AX , EDX:EAX (dim(*src*) × 2)  • *dest* – reg16/32  • *src1* – reg16/32, mem16/32  • *imm*\_*src* – data imediată8/16/32  - *Sofort\_Data*8/16/32 | | |
| 2.1.3 Div, idiv |  | 2.1.3 Div, idiv |
| Instrucţiunea **div** realizează împărțirea numerelor întregi fără semn, iar instrucţiunea **idiv** realizează împărțirea numerelor întregi cu semn. |  | Der **div**-Anweisung führt die Division von ganzen Zahlen ohne Vorzeichen durch, und der **idiv**-Anweisung führt die Division von ganzen Zahlen mit Vorzeichen durch. |
| **div** *src* ; *acc*LO := *acc / src*  ; *acc*HI := *acc* MOD *src*  **idiv** *src* ; *acc*LO := *acc / src*  ; *acc*HI := *acc* MOD *src*   * *src* – reg8/16/32, mem8/16/32 * *acc* – AX, DX:AX, EDX:EAX (2 × dim(*src*)) | | |
| Nu se pot împărţi 2 operanzi de aceeaşi lungime! Indicatorii de condiţie (*flags*) au conţinut imprevizibil după înmulţire si împărţire! |  | Sie können nicht 2 Operanden gleicher Länge teilen! Die Bedingungsindikatoren (Flags) enthielten nach Multiplikation und Division unvorhersehbaren Inhalt! |
| 2.1.4 Neg, inc, dec |  | 2.1.4 Neg, inc, dec |
| Instrucţiunea **neg** calculează opusul unui număr. Instrucţiunea **inc** incrementează conţinutul destinaţiei, iar instrucţiunea **dec** decrementează conţinutul destinaţiei. |  | Die **neg**-Anweisung berechnet das Gegenteil einer Zahl. Der Anweisung **inc** erhöht den Inhalt des Ziels und der Anweisung **dec** verringert den Inhalt des Ziels. |
| **neg** *dest* ; *dest* := -*dest*  **inc** *dest* ; *dest* := *dest* + 1  **dec** *dest* ; *dest* := *dest* - 1   * *dest* – reg16/32, mem16/32 | | |
| Aceste instrucţiuni sunt scurte şi rapide. Ele sunt utile pentru parcurgerea unor şiruri prin incrementarea sau decrementarea adresei. De asemenea, ele sunt utile pentru contorizare (numărare). |  | Diese Anweisungen sind kurz und schnell. Sie sind nützlich, um Zeichenfolgen durch Erhöhen oder Verringern der Adresse auszuführen. Sie sind auch zum Zählen nützlich. |
| 2.2 Operaţii logice pe biţi |  | 2.2 Logische Operationen an Bits |
| Instrucţiunea **AND** este indicată pentru izolarea unui anumit bit sau pentru forţarea anumitor biţi la valoarea 0.  Instrucţiunea **OR** este indicată pentru forţarea anumitor biţi la valoarea 1.  Instrucţiunea **XOR** este indicată pentru schimbarea valorii unor biţi din 0 în 1 sau din 1 în 0. |  | Die **AND (UND)**-Anweisung dient dazu, ein bestimmtes Bit zu isolieren oder bestimmte Bits auf 0 zu setzen.  Die **OR** (**ODER)**-Anweisung wird angezeigt, um bestimmte Bits auf den Wert 1 zu zwingen. Die **XOR**-Anweisung dient zum Ändern des Werts der Bits von 0 auf 1 oder von 1 auf 0. |
| 2.3 Deplasări şi rotiri de biţi |  | 2.3 Bewegungen und Drehungen |
| Instrucţiunile de *deplasare* de biţi se clasifică în:   * Instrucţiuni de deplasare logică: stânga **SHL**, dreapta **SHR** * Instrucţiuni de deplasare aritmetică: stânga **SAL**, dreapta **SAR**   Instrucţiunile de *rotire* a biţilor în cadrul unui operand se clasifică în:   * Instrucţiuni de rotire fără carry: stânga **ROL**, dreapta **ROR** * Instrucţiuni de rotire cu carry: stânga **RCL**, dreapta **R**C**R**   Instrucţiunile de deplasare şi rotire de biţi – tabel – pag.134 curs.    Pentru a defini deplasările şi rotirile să considerăm ca şi configuraţie iniţială un octet X = abcdefgh, unde *a*-*h* sunt cifre binare, *h* este cifra binară de rang 0, *a* este cifra binară de rang 7, iar *k* este valoarea existentă în CF (CF=k).  Atunci avem: |  | Die *Verschiebung* Anweisungen sind unterteilt in:   * Logische VerschiebungsAnweisungen: left (links) **SHL**, right (rechts) **SHR** * Arithmetische VerschiebungsAnweisungen: left (links) **SAL**, right (rechts) **SAR**   Die *Rotation* Anweisungen von Bits in einem Operanden sind in folgende Kategorien unterteilt:   * Anweisungen für Rotation ohne Transport: Linkslauf **ROL**, Rechtslauf **ROR** * Anweisungen für Rotation mit Carry: links **RCL**, rechts **RCR**   Anweisungen zum *Verschiebung* und Rotation von Bits - Tabelle - Seite 134 Kurs.  Um die Verschiebungen und Rotationen zu definieren, betrachten wir als Anfangskonfiguration ein Oktett X = abcdefgh, wobei *a*-*h* Binärziffern sind, *h* die Binärziffer von Rang 0 ist, a die Binärziffer von Rang 7 ist und *k* der in CF vorhandene Wert ist (CF = *k*).  Dann haben wir: |
| **SHL** X,1 ; rezultă (*ergibt sich*) X = bcdefgh0 und CF = a  **SHR** X,1 ; rezultă (*ergibt sich*) X = 0abcdefg und CF = h  **SAL** X,1 ; identisch mit **SHL**  **SAR** X,1 ; rezultă (*ergibt sich*) X = aabcdefg und CF = h  **ROL** X,1 ; rezultă (*ergibt sich*) X = bcdefgha und CF = a  **ROR** X,1 ; rezultă (*ergibt sich*) X = habcdefg und CF = h  **RCL** X,1 ; rezultă (*ergibt sich*) X = bcdefghk und CF = a  **RCR** X,1 ; rezultă (*ergibt sich*) X = kabcdefg und CF = h | | |
| 3. Ramificări, salturi, cicluri |  | 3. Äste, Sprünge, Zyklen |
| 3.1 Saltul necondiţionat |  | 3.1 Der bedingungslose Sprung |
| În această categorie intră instrucţiunile **JMP** (echivalentul instrucţiunii **GOTO** din alte limbaje), **CALL** (apelul de procedură înseamnă transferul controlului din punctul apelului la prima instrucţiune din procedura apelată) şi **RET** (transfer control la prima instrucţiune executabilă de după **CALL**).  **JMP** *operand* ; Salt necondiţionat la adresa  determinată de operand  **CALL** *operand*; Transferă controlul  procedurii determinată de operand  **RET** [*n*] ; Transferă controlul  instrucţiunii de după **CALL** |  | Diese Kategorie umfasst die Anweisung **JMP** (das Äquivalent der **GOTO**-Anweisung in anderen Sprachen), **CALL** (der Prozeduraufruf bedeutet die Übertragung der Steuerung vom Aufrufpunkt zum ersten Anweisung in der aufgerufenen Prozedur) und **RET** (Übertragung der Steuerung zum ersten ausführbaren Anweisung nach **CALL**). **JMP**-Operand; Bedingungsloser Sprung zur Adresse bestimmt von dem Operand  **CALL**-Operand; Übertragen Sie die Kontrolle der Prozedur von den Operand bestimmt  **RET** [*n*]; Übertragen Sie die Kontrolle  nach der **CALL**-Anweisung |
| 3.1.1 Instrucţiunea JMP |  | 3.1.1 Die JMP-Anweisung |
| Instrucţiunea de salt necondiţionat **JMP** are sintaxa : **JMP** *operand*  unde *operand* este o etichetă, un registru sau o variabilă de memorie care conţine o adresă. Efectul ei este transferul necondiţionat al controlului la instrucţiunea care urmează etichetei, la adresa dată de valoarea registrului sau constantei, respectiv la adresa conţinută în variabila de memorie. De exemplu, după execuţia secvenţei  **mov** AX, 1  **jmp** *AdunaDoi*  *AdunaUnu*: **inc** EAX  **jmp** urmare  *AdunaDoi*: **add** EAX, 2  urmare: . . .  registrul AX va conţine valoarea 3. Instrucţiunile **inc** şi **jmp** dintre etichetele *AdunaUnu* şi *AdunaDoi* nu se vor executa, decât dacă se va face salt la *AdunaUnu* de altundeva din program.  Saltul poate fi făcut şi la o adresă memorată într-un registru sau într-o variabilă de memorie. Exemple: |  | Der unbedingte JMP-SprungAnweisung hat die folgende Syntax: **JMP** Operand Dabei ist *Operand* eine Label, ein Register oder eine Speichervariable, die eine Adresse enthält. Seine Wirkung ist die bedingungslose Übergabe der Steuerung an die Anweisung, der auf das Label folgt, an die Adresse, die durch den Wert des Registers bzw. der Konstante an die in der Speichervariablen enthaltene Adresse gegeben ist. Zum Beispiel nach der Sequenzausführung:  **mov** AX, 1  **jmp** *AdunaDoi*  *AdunaUnu*: **inc** EAX  **jmp** urmare (*Ergebnis*)  *AdunaDoi*: **add** EAX, 2  *Ergebnis*: . . .  Das AX-Register enthält den Wert 3. Die Anweisungen **inc** und **jmp** zwischen den Labels *AdunaUnu* und *AdunaDoi* werden nur ausgeführt, wenn von einer anderen Stelle im Programm aus zu *AdunaUnu* gesprungen wird.  Der Sprung kann auch zu einer Adresse erfolgen, die in einem Register oder in einer Speichervariablen gespeichert ist. Beispiele: |
| (1) **mov** EAX, etich  **jmp** EAX ;operand registru  etich: . . .    (2) **segment** data  Salt DD Dest ; Salt := offset Dest  . . .  **segment** cod  . . .  **jmp** [Salt] ; salt NEAR  . . ; operand variabilă de memorie (*Speichervariable*)  Dest : . . . | | |
| Dacă în cazul (1) dorim înlocuirea operandului destinaţie registru cu un operand destinaţie variabilă de memorie, o soluţie posibilă este:  b **resd** 1 ; *reserve double*, 1 dublucuvânt  . . .  **mov** [b], DWORD etich ; *b* := offset etich  **jmp** [b] ;salt NEAR – operand variabilă de  ;memorie JMP DWORD PTR ;DS:[offset\_b] |  | Wenn in Falle (1) der Registrierungszieloperand durch einen variablen Speicherzieloperanden ersetzt werden soll, ist eine mögliche Lösung:  b **resd** 1; *reserve double*, 1 Doppelwort . . .  **mov** [b], DWORD ethic; b: = Offset ethic **jmp** [b]; NEAR springen - Operand  ; Speichervariable JMP DWORD ; PTR DS:[offset\_b] |
| 3.1.2 Instrucţiuni de salt condiţionat şi necondiţionat |  | 3.1.2 Bedingte und unbedingte SprungAnweisungen |
| Următoarele instrucţiuni au ca efect realizarea unui salt în cadrul programului doar dacă anumite condiţii sunt îndeplinite.  **Jcc** - salt dacă condiția ‘cc’ este îndeplinită; în caz contrar se trece la instrucțiunea următoare  Sintaxa: **Jcc** <eticheta> , unde:  – <eticheta> - se traduce printr-o distanţă relativă pe 8 biţi  – condiţia este dată de starea unui sau a unor indicatoare de condiţie (*flag*-uri): CF, ZF, SF, PF, OF  – pentru aceeași condiție pot exista mnemonici diferite (ex: JZ, JE)  – Atenție: la 8086/286 salturile pot fi doar in intervalul -128 .. +127;  – De la ‘386 salturile se pot face oriunde în interiorul unui segment |  | Die folgenden Anweisungen bewirken nur dann einen Sprung innerhalb des Programms, wenn bestimmte Bedingungen erfüllt sind.  **Jcc** - Sprung, wenn die Bedingung 'cc' erfüllt ist; Fahren Sie andernfalls mit der folgenden Anweisung fort  Syntax: **Jcc** <label>, wobei:  - <label> - übersetzt in einen relativen Abstand von 8 Bits  - Die Bedingung wird durch den Status eines oder mehrerer Bedingungsindikatoren (*Flags*) angegeben: CF, ZF, SF, PF, OF  - für den gleichen Zustand kann es verschiedene Mnemoniken geben (zB: JZ, JE) - Achtung: bei 8086/286 können Sprünge nur im Bereich -128 .. +127 liegen;  - Ab '386 können Sprünge an beliebiger Stelle innerhalb eines Segments ausgeführt werden |
| **Tabelul 2.** *Instrucțiuni de salt condiționat din perspectiva indicatorilor de condiție (*Anweisungen für bedingte Sprünge aus der Perspektive der Bedingungsindikatoren*)*   |  |  |  | | --- | --- | --- | | **Instrucţiune (*Anweisung*)** | **Condiţie (*Bedingung*)** | **Alias** | | JC | CF = 1 | JB, JNAE | | JNC | CF = 0 | JNB, JAE | | JZ | ZF = 1 | JE | | JNZ | ZF = 0 |  | | JS | SF = 1 |  | | JNS | SF = 0 |  | | JO | OF = l | JPE | | JNO | OF = 0 | JP | | JP | PF = 1 | JPO | | JNP | PF = 0 | JNP |   **Tabelul 3.** *Instrucțiuni de salt condiționat – comparare numere fără semn (*Bedingte Sprunganweisungen – Vergleich von Zahlen ohne Vorzeichen*)*   |  |  |  |  | | --- | --- | --- | --- | | **Instrucţiune (*Anweisung*)** | **Condiţie (*Bedingung*)** | **Indicator** | **Alias** | | JA | > | CF = 0; ZF = 0 | JNBE | | JAE | >= | CF = 0 | JNB, JNC | | JB | < | CF = 1 | JNAE, JC | | JBE | <= | CF = l oder ZF = l | JNA | | JE | = | ZF = 1 | JZ | | JNE | != | ZF = 0 | JNZ |   **Tabelul 4.** *Instrucțiuni de salt condiționat – comparare numere cu semn (*Bedingte Sprunganweisungen – Vergleich von Zahlen mit Vorzeichen*)*   |  |  |  |  | | --- | --- | --- | --- | | **Instrucţiune (*Anweisung*)** | **Condiţie (*Bedingung*)** | **Indicator** | **Alias** | | JG | > | SF = OF sau ZF = 0 | JNLE | | JGE | >= | SF = OF | JNL | | JL | < | SF != OF | JNGE | | JLE | <= | SF != OF sau ZF = 1 | JNG | | JE | = | ZF = 1 | JZ | | JNE | != | ZF = 0 | JNZ | | | |
| În Tabelul 2, Tabelul 3 şi Tabelul 4 (vezi şi tabelul de la pagina 146 din cursul tipărit) se prezintă instrucţiunile de salt condiţionat împreună cu semnificaţia lor şi cu precizarea valorilor *flag*-urilor în urma cărora se execută salturile respective. Pentru toate instrucţiunile de salt sintaxa este aceeaşi:  *<instrucţiune\_de\_salt> etichetă*  Semnificaţia instrucţiunilor de salt condiţionat este dată sub forma ***„salt dacă operand1 <<relaţie>> faţă de operand2”*** (unde cei doi operanzi sunt obiectul unei instrucţiuni anterioare **CMP** sau **SUB**) sau referitor la valoarea concretă setată pentru un anumit *flag*.  Când se compară două numere cu semn se folosesc termenii **„less than”** (mai mic decât) şi „**greater than”** (mai mare decât), iar când se compară două numere fără semn se folosesc termenii „*below*” (inferior, sub) şi respectiv „*above*” (superior, deasupra, peste). |  | In Tabelle 2, Tabelle 3 und Tabelle 4 (siehe auch Tabelle auf Seite 146 des gedruckten Kurses) sind die Anweisungen für bedingte Sprünge zusammen mit ihrer Bedeutung und den *Flag*-Werten aufgeführt, nach denen die jeweiligen Sprünge ausgeführt werden. Für alle Sprungbefehle gilt die gleiche Syntax:  *<SprungAnweisung> Label*  Die Bedeutung der bedingten Sprunganweisungen wird in Form von ***„Sprung wenn Operand1 << relation >> in Richtung Operand2“*** (wobei die beiden Operanden das Objekt einer vorherigen **CMP**- oder **SUB**-Anweisung sind) oder unter Bezugnahme auf den konkreten Wert angegeben, der für ein bestimmtes Flag festgelegt wurde.  Wenn Sie zwei Zahlen mit einem Vorzeichen vergleichen, werden die Begriffe **„less than”** („kleiner als“) und **„greater than”** („größer als“) verwendet, und wenn Sie zwei Zahlen ohne Vorzeichen vergleichen, werden die Begriffe „*below*” („unten“) und verwendet jeweils „*above*” („oben“). |
| 3.1.3 Comparaţii între operanzi |  | 3.1.3 Vergleiche zwischen Operanden |
| Instrucţiunile de salt condiţionat se folosesc de obicei în combinaţie cu instrucţiuni de comparare. De aceea, semnificaţiile instrucţiunilor de salt rezultă din semnificaţia operanzilor unei instrucţiuni de comparare. În afara testului de egalitate pe care îl poate efectua o instrucţiune **CMP** este de multe ori necesară determinarea relaţiei de ordine dintre două valori. De exemplu, se pune întrebarea: numărul 11111111b (= FFh = 255 = -1) este mai mare decât 00000000b(= 0h = 0)? Răspunsul poate fi şi da şi nu! Dacă cele două numere sunt considerate fără semn, atunci primul are valoarea 255 şi este evident mai mare decât 0. Dacă cele două numere sunt considerate cu semn, atunci primul are valoarea -1 şi este mai mic decât 0. |  | Bedingte SprungAnweisungen werden normalerweise in Kombination mit VergleichsAnweisungen verwendet. Die Bedeutungen der Sprungbefehle ergeben sich daher aus der Bedeutung der Operatoren eines VergleichsAnweisungs. Abgesehen von der Gleichheitsprüfung, die ein **CMP**-Anweisung durchführen kann, ist es häufig erforderlich, die Ordnungsbeziehung zwischen zwei Werten zu bestimmen. Beispielsweise wird die Frage gestellt: Ist die Nummer 11111111b (= FFh = 255 = -1) größer als 00000000b (= 0h = 0)? Die Antwort kann ja und nein sein! Wenn die beiden Zahlen als vorzeichenlos betrachtet werden, ist die erste 255 und offensichtlich größer als 0. Wenn die beiden Zahlen als Vorzeichen betrachtet werden, ist die erste -1 und kleiner als 0. |
| |  |  |  | | --- | --- | --- | | **CMP** *d,s* | comparaţia valorilor operanzilor (nu modifică operanzii) (execuţie fictivă *d - s*)  *Vergleich der Werte der Operanden (ändert die Operanden nicht) (fiktive Ausführung* d – s, *Subtraktion)* | OF, SF, ZF, AF, PF şi CF | | **TEST** *d,s* | execuţie fictivă *d* **AND** *s*  *(fiktive Ausführung d* **UND** *s)* | OF = 0, CF = 0  SF, ZF, PF – modificaţi (*modifizierten*), AF – nedefinit (*undefiniert*) | | | |
| Instrucţiunea **CMP** nu face distincţie între cele două situaţii, deoarece adunarea şi scăderea se efectuează întotdeauna la fel (adunând sau scăzând configuraţii binare) indiferent de semnul (interpretarea) acestor configuraţii. Ca urmare, nu este vorba de a interpreta cu semn sau fără semn ***operanzii*** scăderii fictive *d-s*, ci ***rezultatul*** final al acesteia! Rolul de a interpreta în mod diferit (cu semn sau fără semn) rezultatul final al comparaţiei revine diverselor instrucţiuni de salt condiţionat. |  | Die **CMP**-Anweisung unterscheidet nicht zwischen den beiden Situationen, da die Addition und Subtraktion unabhängig vom Vorzeichen (Interpretation) dieser Konfigurationen immer gleich ausgeführt werden (durch Addieren oder Subtrahieren von Binärkonfigurationen). Es geht also nicht darum, ***die*** ***Operanden*** der fiktiven Subtraktion *d* - *s* mit oder ohne Vorzeichen zu interpretieren, sondern um ***das Endergebnis***! Die Rolle der unterschiedlichen Interpretation (mit oder ohne Vorzeichen) des Endergebnisses des Vergleichs liegt in den verschiedenen bedingten SprungAnweisungen. |
| 3.1.4 Instrucţiuni de ciclare |  | 3.1.4 SchleifAnweisungen |
| Instrucţiunea **JCXZ** (**JECXZ)** permite realizarea unui salt dacă registrul CX (respectiv ECX) are valoarea 0. Ea se foloseşte înaintea unei instrucţiuni de buclare (**LOOP**) pentru a preîntâmpina execuția de 65.535 ori a buclei, în cazul în care CX=0.  O altă instrucţiune de buclare este **LOOP**, având sintaxa:  **LOOP** <eticheta>  În cadrul acestei instrucţiuni se efectuează:  ECX = ECX - 1  **if** (ECX != 0) „salt la <*eticheta*>”  **else** „continuă cu instrucțiunea următoare”  Saltul este „scurt” (max. 127 octeţi – atenţie deci la „distanţa” dintre **LOOP** şi etichetă!).  ECX este folosit implicit pentru contorizarea ciclurilor executate.  Dintre instrucţiunile de salt necondiţionat, prezentăm **LOOPZ** şi **LOOPE** (***LOOP*** *while* ***E****qual*) – instrucţiuni de buclare.  Sintaxa:  **LOOPZ** | **LOOPE** <eticheta>  La fel ca în cazul instrucţiunii **LOOP**, acţiunea lor este următoarea:  ECX = ECX - 1  **if** ((ECX != 0) si (ZF = 1) „salt la *eticheta*”  **else** „continuă”  Alte instrucţiuni de buclare sunt **LOOPNZ** şi **LOOPNE**. Sintaxa lor:  **LOOPNZ** | **LOOPNE** <eticheta>  Acţiunea lor:  ECX = ECX - 1  **if** ((ECX!=0) şi (ZF!=1) „salt la *eticheta*”  **else** „continuă”  Nici una dintre instrucţiunile de ciclare prezentate nu afectează *flag*-urile. |  | Die Anweisung **JCXZ** (**JECXZ**) ermöglicht einen Sprung, wenn das CX-Register (bzw. ECX) auf 0 gesetzt ist. Sie wird vor einem SchleifAnweisung (**LOOP**) verwendet, um die Ausführung einer 65.535-fachen Schleife zu verhindern, wenn CX = 0 ist.  Ein weiterer SchleifAnweisung ist **LOOP** mit der Syntax:  **LOOP** <Label>  Innerhalb dieser Anweisung werden ausgeführt: ECX = ECX – 1  **if** (ECX! = 0) „springe zu <*Label*>“ **else** „fahren Sie mit der folgenden Anweisung fort“  Der Sprung ist „kurz“ (max. 127 Byte - also „Abstand“ zwischen **LOOP** und Label beachten!). ECX wird standardmäßig zum Zählen der ausgeführten Zyklen verwendet.  Aus den bedingungslosen SprungAnweisungen präsentieren wir **LOOPZ**- und **LOOPE** (***LOOP*** *while Equal*) - SchleifenAnweisungen. Syntax:  **LOOPZ** | **LOOPE** <Label>  Wie bei der **LOOP**-Anweisung ist ihre Aktion wie folgt:  ECX = ECX - 1  **if** ((ECX! = 0) und (ZF = 1) „zum *Label* springen“ **else** „weiter“  Andere SchleifenAnweisungen sind **LOOPNZ** und **LOOPNE**. Ihre Syntax:  **LOOPNZ** | **LOOPNE** <Label>  Ihre Aktion:  ECX = ECX – 1  **if** ((ECX! = 0) und (ZF! = 1) „zum *Label* springen“ **else** „weiter“  Keine der vorgestellten SchleifenAnweisungen wirkt sich auf die *Flaggen* aus. |
| 3.2 Instrucţiunile CALL şi RET |  | 3.2 CALL- und RET-Anweisungen |
| Apelul unei proceduri se face cu ajutorul instrucţiunii **CALL**, acesta putând fi *apel direct* sau *apel indirect*. Apelul direct are sintaxa:  **CALL** *operand*  Asemănător instrucţiunii **JMP** şi instrucţiunea **CALL** transferă controlul la adresa desemnată de operand. În plus faţă de aceasta, înainte de a face saltul, instrucţiunea **CALL** salvează în stivă adresa următoarei instrucţiuni de după **CALL** (adresa de revenire). Cu alte cuvinte, avem echivalenţa: |  | Der Aufruf einer Prozedur erfolgt mit der Anweisung **CALL**, die ein *direkter* oder ein *indirekter Aufruf* sein kann. Der direkte Aufruf hat die Syntax:  **CALL** Operand  Wie die **JMP**-Anweisung die **CALL**- Anweisung überträgt die Steuerung an die vom Operanden angegebene Adresse. Zusätzlich speichert die **CALL**-Anweisung vor dem Sprung die Adresse des nächsten Anweisung nach dem **CALL** (Rücksprungadresse) im Stapel. Mit anderen Worten, wir haben die Äquivalenz: |
| **CALL** operand **push** A  A: . . . ⇔ **jmp** operand | | |
| Terminarea execuţiei secvenţei apelate este marcată de întâlnirea unei instrucţiuni **RET**. Aceasta preia din stivă adresa de revenire depusă acolo de **CALL**, predând controlul la instrucţiunea de la această adresă. Sintaxa instrucţiunii **RET** este:  **RET** [*n*]  unde *n* este un parametru opţional. El indică eliberarea din stivă a *n* octeţi aflaţi sub adresa de revenire.  Instrucţiunea **RET** poate fi ilustrată prin echivalenţa: |  | Der Abschluss der Ausführung der aufgerufenen Sequenz wird durch das Treffen eines **RET**-Anweisungs gekennzeichnet. Es übernimmt vom Stapel die dort hinterlegte Rücksprungadresse von **CALL** und übergibt die Steuerung an die Anweisung von dieser Adresse. Die Syntax der **RET**-Anweisung lautet:  **RET** [n]  Dabei ist *n* ein optionaler Parameter. Es zeigt die Freigabe von *n* Bytes unter der Rücksprungadresse an.  Die **RET**-Anweisung kann durch die Äquivalenz veranschaulicht werden: |
| B dd ?  **RET** n . . .  (revenire (*Rückkehr*) near) ⇔ **pop** [B]  **add** ESP,[n]  **jmp** [B] | | |
| De cele mai multe ori, instrucţiunile **CALL** şi **RET** apar în următorul context: |  | In den meisten Fällen werden die Anweisungen **CALL** und **RET** in folgendem Kontext angezeigt: |
| etichetă\_procedură (*Label\_Prozedur*):  . . .  **ret** n  . . .  **CALL** etichetă\_procedură (*Label\_Prozedur*) | | |
| Instrucţiunea **CALL** poate de asemenea prelua adresa de transfer dintr-un registru sau dintr-o variabilă de memorie. Un asemenea gen de apel este denumit *apel indirect*. Exemple:  **call** EBX ; adresă preluată din  registru  **call** [vptr] ;adresă preluată din memorie  (similar cu apelul funcţiei *printf*)  Rezumând, operandul destinaţie al unei instrucţiuni **CALL** poate fi:  - numele unei proceduri  - numele unui registru în care se află o adresă  - o adresă de memorie |  | Die **CALL**-Anweisung kann auch die Übertragungsadresse aus einem Register oder einer Speichervariablen abrufen. Diese Art von Anruf wird *indirekter Anruf* genannt. Beispiele:  **call** EBX ;Adresse entnommen aus  ; ein Register  **call** [vptr]; Adresse aus dem Speicher  ; genommen (ähnlich wie Funktion ; *printf* aufrufen)  Zusammenfassend kann der Zieloperand einer **CALL**-Anweisung sein:   * der Name einer Prozedur * der Name eines Registers, in dem sich eine Adresse befindet * eine Speicheradresse |
| 4. Instrucţiuni pe şiruri |  | 4. String Anweisungen |
| 4.1 Generalităţi privind instrucţiunile pe şiruri |  | **4.1 Allgemeines zu String-Anweisungen** |
| Instrucţiunile pe şiruri permit manipularea unui bloc de date printr-o singură instrucţiune. Sunt singurele instrucțiuni care permit transfer memorie-memorie sau memorie-port de intrare / ieşire.  Instrucţiunile pe şiruri sunt puternice, ele având ca efect simultan (eventual repetat) atât accesarea memoriei cât şi incrementarea sau decrementarea unor registre pointer. Se folosesc pentru manipularea şirurilor de octeţi sau cuvinte, fiind mai scurte (ca şi cod rezultat) şi mai rapide în execuţie decât combinaţiile echivalente de instrucţiuni **MOV**, **INC** şi **LOOP**.  Instrucțiunile pe şiruri folosesc operanzi impliciți:   * DS:ESI – adresa elementului din şirul sursă * ES:EDI – adresa elementului din şirul destinație * ECX – contor * AL/AX – registru acumulator * incrementarea sau decrementarea automată a registrelor index (ESI, EDI) în funcţie de starea *flag*-ului DF (*Direction Flag*) (0 = incrementare, 1 = decrementare) * decrementarea registrului ECX   Un şir în sensul 8086 este caracterizat de următoarele atribute:   1. tipul elementelor (octeţi sau cuvinte). 2. adresa primului element din şir. 3. direcţia de parcurgere (de la adrese mici spre adrese mari sau invers). 4. numărul de elemente.   Din punct de vedere al instrucţiunilor pe şiruri, un şir poate fi şir sursă sau şir destinaţie.  Instrucţiunile pe şiruri sunt în număr de 10 şi se împart în trei categorii:   * instrucţiuni care folosesc un şir sursă şi un şir destinaţie (**MOVSB**, **MOVSW**, **CMPSB**, **CMPSW**). * instrucţiuni care folosesc numai un şir sursă (**LODSB**, **LODSW**). * instrucţiuni care folosesc numai un şir destinaţie (**STOSB**, **STOSW**, **SCASB**, **SCASW**).   Fiecare dintre aceste instrucţiuni pretinde să-i fie pregătite în prealabil caracteristicile şirurilor cu care operează.  a) *Tipul* este indicat prin ultima literă a numelui instrucţiunii: B pentru elemente octeţi şi W pentru elemente cuvinte. Instrucţiunile care folosesc două şiruri presupun că ambele şiruri sunt de acelaşi tip.  b) *Adresa* *primului element dintr-un şir* este o adresă FAR memorată în regiştri, astfel:   * în DS:SI pentru şirurile care sunt sursă; * în ES:DI pentru şirurile care sunt destinaţie;   c) Direcţia de parcurgere este indicată de flagul DF, astfel:   * DF = 0 impune ca ordinea de parcurgere să fie de la adrese mici spre adrese mari. În acest caz, adresa primului element este adresa cea mai mică din şir. * DF = 1 impune ca ordinea de parcurgere să fie de la adrese mari spre adrese mici. În acest caz, adresa primului element este adresa cea mai mare din şir.   Poziţionarea prealabilă a lui DF se face folosind instrucţiunile **CLD** sau **STD**. Instrucţiunile care folosesc două şiruri presupun că ambele şiruri sunt parcurse în aceeaşi direcţie.  d) Numărul de elemente, atunci când se doreşte a fi exploatat, trebuie trecut în registrul CX. |  | String-Anweisungen ermöglichen die Manipulation eines Datenblocks durch eine einzelne Anweisung. Sie sind die einzigen Anweisungen, die eine Speicher-Speicher-Übertragung oder eine Speicher-Port-Eingabe / Ausgabe ermöglichen.  String-Anweisungen sind leistungsfähig und bewirken den gleichzeitigen (möglicherweise wiederholten) Zugriff auf den Speicher sowie das Inkrementieren oder Dekrementieren von Zeigerregistern. Sie werden für die Verarbeitung von Byte- oder Wortfolgen verwendet. Sie sind kürzer (als Ergebniscode) und werden schneller ausgeführt als die entsprechenden Kombinationen von **MOV**-, **INC**- und **LOOP**-Anweisungen. String-Anweisungen verwenden implizite Operanden:   * DS: ESI - Die Adresse des Elements in der Quellzeichenfolge * ES: EDI - Die Adresse des Elements in der Zielzeichenfolge * ECX – Zähler * AL / AX – Batterieregister * Automatisches Inkrementieren oder Dekrementieren der Indexregister (ESI, EDI) in Abhängigkeit vom Status des DF-Flags (*Direction Flag*) (0 = Inkrementieren, 1 = Dekrementieren) * Dekrementierung des ECX-Registers   Ein String im Sinne von 8086 zeichnet sich durch folgende Attribute aus:   1. die Art der Elemente (Bytes oder Wörter). 2. die Adresse des ersten Elements in der Zeichenfolge. 3. die Fahrtrichtung (von kleinen Adressen zu großen Adressen oder umgekehrt). 4. die Anzahl der Elemente.   Aus der Sicht der String-Anweisungen kann eine Zeichenfolge eine Quellzeichenfolge oder eine Zielzeichenfolge sein.  Die Anweisungen in Zeilen befinden sich in Nummer 10 und lassen sich in drei Kategorien einteilen:   * Anweisungen, die eine Quell- und eine Zielzeichenfolge verwenden (**MOVSB**, **MOVSW**, **CMPSB**, **CMPSW**). * Anweisungen, die nur eine Quellzeichenfolge verwenden (**LODSB**, **LODSW**). * Anweisungen, die nur eine Zielzeichenfolge verwenden (**STOSB**, **STOSW**, **SCASB**, **SCASW**).   Jede dieser Anweisungen verlangt, dass die Eigenschaften der Saiten, mit denen sie arbeiten, im Voraus vorbereitet werden.  a) *Der Typ* wird durch den letzten Buchstaben des Befehlsnamens angegeben: B für Byteelemente und W für Wortelemente. Anweisungen, die zwei Zeichenfolgen verwenden, setzen voraus, dass beide Zeichenfolgen vom gleichen Typ sind. b) *Die Adresse* *des ersten Elements in einer Zeichenfolge* ist eine FAR-Adresse, die wie folgt in den Registern gespeichert ist:  - in DS:SI für die Zeichenfolgen, die als Quelle dienen;  - in ES:DI für die Zeichenfolgen, die das Ziel sind;  c) Die Fahrtrichtung wird durch die DF-Flagge wie folgt angezeigt:   * DF = 0 erfordert eine Fahrreihenfolge von kleinen zu großen Adressen. In diesem Fall ist die Adresse des ersten Elements die kleinste Adresse in der Zeichenfolge. * DF = 1 setzt voraus, dass die Fahrreihenfolge von großen zu kleinen Adressen reicht. In diesem Fall ist die Adresse des ersten Elements die größte Adresse in der Zeichenfolge.   Die Vorpositionierung des DF erfolgt mit den Anweisungen **CLD** oder **STD**. Anweisungen, die zwei Zeilen verwenden, setzen voraus, dass sich beide Zeilen in dieselbe Richtung bewegen. d) Die Anzahl der Elemente muss, falls gewünscht, in das CX-Register eingetragen werden. |
| 4.2 Instrucţiuni pe şiruri pentru transferul de date |  | 4.2 String-Anweisungen für die Datenübertragung |
| Aceste instrucţiuni sunt asemănătoare instrucţiunii **MOV**, dar ele realizează mai mult şi operează mai rapid.  Instrucţiunea **LODS** se prezintă sub două forme: **LODSB** şi **LODSW**. Ea încarcă un octet sau respectiv un cuvânt din memorie în registrul acumulator. |  | Diese Anweisungen ähneln der **MOV**-Anweisung, leisten jedoch mehr und arbeiten schneller.  Die **LODS**-Anweisung hat zwei Formen: **LODSB** und **LODSW**. Es lädt ein Byte oder ein Wort aus dem Speicher in das Batterieregister. |
| **Tabelul 5.** *Instrucțiuni pe şiruri pentru transferul de date (*String-Anweisungen für die Datenübertragung*)*   |  |  | | --- | --- | | **LODSB** | AL ← <DS:SI> **if** DF = 0 inc(SI) **else** dec(SI) | | **LODSW** | AX ← <DS:SI> **if** DF = 0 SI ← SI + 2 **else** SI ← SI - 2 | | **STOSB** | <ES:DI> ← AL **if** DF = 0 inc(DI) **else** dec(DI) | | **STOSW** | <ES:DI> ← AX **if** DF = 0 DI ← DI+2 **else** DI ← DI - 2 | | **MOVSB** | <ES:DI> ← <DS:SI>; **if** DF = 0 (inc(ST); inc(DT)} **else** {dec(SI); dec(DI)} | | **MOVSW** | <ES:D1> ← <DS:SI>;**if** DF = 0 {SI ← SI + 2; DI ← DI + 2} **else** {SI ← SI - 2;  DI ← DI-2} | | | |
| Instrucţiunea **LODSB** încarcă octetul de adresă DS:ESI în AL şi apoi incrementează sau decrementează ESI, aceasta depinzând de starea *flag*-ului DF (*Direction Flag*): dacă DF=0 (valoare care se poate seta după cum am văzut prin instrucţiunea **CLD**) atunci ESI este incrementat, iar dacă DF=1 (setarea acestuia cu 1 făcându-se cu instrucţiunea **STD**) ESI este decrementat. Această regulă impusă de valoarea din DF este valabilă pentru toate instrucţiunile pe şiruri ce afectează regiştri pointer. De exemplu: |  | Die **LODSB**-Anweisung lädt das Byte der Adresse DS:ESI in AL und erhöht oder verringert dann ESI, abhängig vom Zustand des DF-Flags (*Direction Flag*): wenn DF = 0 (Wert, der so eingestellt werden kann, wie wir es durch den **CLD**-Anweisung gesehen haben) ESI wird inkrementiert, und wenn DF = 1 ist (mit **STD**-Anweisung auf 1 setzen), wird ESI dekrementiert. Diese Regel, die durch den Wert im DF erzwungen wird, gilt für alle Anweisungen für Zeichenfolgen, die Zeigerregister betreffen. Zum Beispiel: |
| **cld**  **mov** SI,0  **lodsb** | | |
| va încărca AL cu conţinutul octetului de deplasament 0 din cadrul segmentului de date şi apoi se va incrementa ESI cu 1, acţiuni echivalente cu |  | wird AL mit dem Inhalt des Verschiebungsbytes 0 im Datensegment laden und dann ESI um 1 erhöhen, Aktionen äquivalent zu |
| **mov** ESI, 0  **mov** AL, [ESI]  **inc** ESI | | |
| Instrucţiunea **LODSB** este însă mai rapidă şi cu 2 octeţi mai scurtă decât echivalentul |  | Der **LODSB**-Anweisung ist jedoch schneller und 2 Byte kürzer als der entsprechende |
| **mov** AL, [ESI]  **inc** ESI | | |
| Instrucţiunea **LODSW** încarcă în AX cuvântul adresat de DS:ESI, incrementând sau decrementând apoi ESI cu 2 (deoarece este vorba de valori reprezentate pe cuvânt).  Instrucţiunea **STOS** este complementara instrucţiunii **LODS**, ea transferând valoarea octet sau cuvânt din acumulator la locaţia de memorie de adresă ES:DI, incrementând sau decrementând apoi corespunzător pe DI. Şi instrucţiunea **STOS** se prezintă sub două forme: **STOSB** şi **STOSW**.  Instrucţiunea **STOSB** copiază octetul din AL la octetul de adresă ES:EDI, incrementând sau decrementând EDI în funcţie de valoarea din DF.  Instrucţiunea **STOSW** este asemănătoare, copiind valoarea cuvântului din AX în cuvântul adresat de ES:EDI, incrementând sau decrementând apoi EDI cu 2.  Instrucţiunile **LODS** şi **STOS** funcţionează eficient împreună pentru copierea de şiruri. Subrutina COPIERE de mai jos realizează copierea şirului terminat cu 0 care începe la DS:ESI în şirul ce începe la adresa ES:EDI:  ; Subrutină pentru copierea unui şir  ; terminat cu 0 în altul  ; Intrări: adresa de început a şirului sursă  ; DS:SI  ; adresa de început a şirului destinaţie ES:DI  ; Ieşiri: -  ; Regiştri afectaţi: AL, SI, DI  Copiere **PROC**  **cld** ;parcurgerea se va face crescător ; deci se va impune incrementarea  iar: **lodsb** ; preia caracterul sursă  **stosb** ; memorează la destinaţie  **cmp** AL,0; a fost 0?  **jnz** iar ; dacă nu, se continuă  **ret** ; dacă da, se revine din  procedură  Copiere **ENDP**  O modalitate mai eficientă de transfer a unui octet sau cuvânt dintr-o locaţie de memorie în alta este prin folosirea instrucţiunii **MOVS**. Aceasta este o combinaţie a instrucţiunilor **LODS** şi **STOS**, ea preluând octetul (**MOVSB**) sau cuvântul (**MOVSW**) de la DS:ESI şi depunând această valoare la adresa ES:EDI. Nu este folosit ca intermediar nici un registru deci nici o valoare a acestora nu va fi afectată.  Folosind **MOVSW**, ciclul de mai sus devine  Bucla: **movsw**  **loop** Bucla |  | Die **LODSW**-Anweisung lädt das von DS:ESI adressierte Wort in die AX und erhöht oder verringert dann ESI um 2 (da es sich um die durch das Wort dargestellten Werte handelt).  Die **STOS**-Anweisung ist komplementär zur **LODS**-Anweisung, die den Byte- oder Wortwert von der Akkumulator an den Speicherort der ES: DI-Adresse überträgt und den DI dann entsprechend inkrementiert oder dekrementiert. Und die **STOS**-Anweisung gibt es in zwei Formen: **STOSB** und **STOSW**.  Die **STOSB**-Anweisung kopiert das Byte von AL in das Adressbyte ES:EDI und erhöht oder verringert EDI abhängig vom Wert in DF. Die **STOSW**-Anweisung ist ähnlich: Sie kopiert den Wortwert von AX in das von ES:EDI adressierte Wort und erhöht oder verringert dann EDI um 2.  Die Anweisungen **LODS** und **STOS** arbeiten beim Kopieren von Zeichenfolgen effizient zusammen. Das folgende Unterprogramm COPIERE kopiert die Zeichenfolge, die mit 0 beginnt und mit DS:ESI beginnt, in die Zeichenfolge, die mit der Adresse ES:EDI beginnt: ; Unterprogramm zum Kopieren eines Strings ; endet mit 0 in einem anderen  ; Eingaben: Die Startadresse der Quellzeichenfolge  ; DS: SI  ; die Startadresse des Zielstrings ES: DI  ; Ausgänge: -  ; Betroffene Register: AL, SI, DI  Copiere **PROC**  **cld** ; Die Reise wird inkrementell  ; durchgeführt, sodass eine Erhöhung ; erforderlich ist  wieder: **lodsb** ; übernimmt den Quellcharakter  **stosb** ; merkt zu Ziel  **cmp** AL,0; es war 0?  **jnz** wieder ; wenn nicht, fahrt fort  **ret** ; wenn ja, kommt von der ; Prozedur zurück  Copiere **ENDP**  Eine effizientere Möglichkeit, ein Byte oder Wort von einem Speicherort zu einem anderen zu übertragen, ist die Verwendung der **MOVS**-Anweisung. Dies ist eine Kombination der Anweisungen **LODS** und **STOS**, wobei das Byte (**MOVSB**) oder das Wort (**MOVSW)** von DS:ESI genommen und dieser Wert an die Adresse ES:EDI gesendet wird. Es wird kein Register als Vermittler verwendet, sodass kein Wert beeinträchtigt wird.  Mit **MOVSW** wird der obige Zyklus  Bucla: **movsw**  **loop** Bucla |
| 4.3 Instrucţiuni pe șiruri pentru consultarea şi compararea datelor |  | **4.3 String-Anweisungen zum Abfragen und Vergleichen von Daten** |
| Instrucţiunea **SCAS** (care are şi ea două variante: **SCASB** şi **SCASW**) caută în memorie o anumită valoare particulară (octet sau respectiv cuvânt).  Instrucţiunea **SCASB** compară valoarea din AL cu valoarea octet adresată de ES:EDI, setând *flag*-urile în acord cu rezultatele comparării (la fel ca şi o instrucţiune **CMP**). După aceea, EDI este incrementat sau decrementat. |  | Die **SCAS**-Anweisung (die auch zwei Varianten hat: **SCASB** und **SCASW**) sucht nach einem bestimmten Wert im Speicher (Byte bzw. Wort).  Die **SCASB**-Anweisung vergleicht den Wert in AL mit dem von ES:EDI adressierten Bytewert und setzt die *Flags* entsprechend den Vergleichsergebnissen (sowie einer **CMP**-Anweisung). Danach wird EDI erhöht oder verringert. |
| **Tabelul 6.** *Instrucțiuni pe şiruri pentru consultarea şi compararea datelor (*String-Anweisungen zum Abfragen und Vergleichen von Daten*)*   |  |  |  | | --- | --- | --- | | **SCASB** | CMP AL,<ES:DI> ; **if** DF = 0 inc(DI) **else** dec(DI) | **OF, SF, ZF , AF, PF, CF** | | **SCASW** | CMP AX,<ES:DI> ; **if** DF = 0 DI ← DI + 2 **else** DI ← DI - 2 | **OF, SF, ZF , AF, PF, CF** | | **CMPSB** | CMP <DS:SI>,<ES:DI>; **if** DF = 0 {inc(SI); inc(DI)} **else**  ; {dec(SI); dec(DI)} | **OF, SF, ZF , AF, PF, CF** | | **CMPSW** | CMP <DS:SI>,<ES:DI>; **if** DI=0 {SI ← SI + 2; DI ← DI + 2}  ; **else** {SK-SI-2; **DK-DI-2)** | **OF, SF, ZF ,AF, PF, CF** | | | |
| Instrucţiunile pe şiruri NU setează *flag*-urile în urma acţiunii asupra regiştrilor ESI, EDI sau ECX. Instrucţiunile **LODS**, **STOS** şi **MOVS** nu afectează nici un *flag*, iar **SCAS** şi **CMPS** modifică *flag*-urile doar ca rezultat al comparaţiilor pe care le efectuează.  Instrucţiunea **SCASW** compară conţinutul registrului AX cu cuvântul de adresă ES:EDI incrementând sau decrementând EDI cu 2, în funcţie de valoarea din DF. Secvenţa de mai jos utilizează instrucţiunea **SCASW** cu prefixul **REPE** pentru a căuta ultima valoare nenulă dintr-un vector de întregi: |  | String-Anweisungen setzen KEINE *Flags* nach einer Aktion auf ESI-, EDI- oder ECX-Register. Die Anweisungen **LODS**, **STOS** und **MOVS** wirken sich nicht auf Flags aus, und **SCAS** und **CMPS** ändern die Flags nur aufgrund der durchgeführten Vergleiche.  Die **SCASW**-Anweisung vergleicht den Inhalt des AX-Registers mit dem Adresswort ES:EDI, indem sie EDI je nach DF-Wert um 2 erhöht oder verringert. In der folgenden Sequenz wird die **SCASW**-Anweisung mit dem Präfix **REPE** verwendet, um in einem Ganzzahlvektor nach dem letzten Wert ungleich Null zu suchen: |
| **mov** AX, SEG Tablou  **mov** ES, AX  **mov** DI, OFFSET Tablou + ((NrElem - 1 ) \* 2) ; ES:DI punctează astfel spre ultimul ; element al tabloului  *; ES:DI verweist also auf das letzte Element des*  *; Arrays*  **mov** CX, NrElem  **sub** AX, AX ; pune 0 în AX pentru a căuta un element nenul  *; setze 0 in AX, um nach einem Nicht-Null-Element zu suchen*  **std** ;căutarea începe de la sfârşit  *; Die Suche beginnt am Ende*  **repe** **scasw** ; se repetă căutarea până la primul element nenul sau până la  ; epuizarea elementelor tabloului  *; Die Suche wird wiederholt, bis das erste Nicht-Null-Element ; oder die Elemente des Arrays erschöpft sind*  **jne** Gasit ;dacă se ajunge aici, atunci tabloul conţine numai elemente nule  *; Wenn Sie hierher kommen, enthält das Array nur Null-Elemente*  Gasit: **inc** di  **inc** di ; se actualizează DI pentru a puncta spre elementul găsit  *; Die DI wird aktualisiert, um auf das gefundene Objekt zu verweisen* | | |
| Instrucţiunea **CMPS** are ca rol efectuarea de comparări de şiruri de octeţi sau cuvinte. Execuţia unei instrucţiuni **CMPS** are ca efect compararea locaţiilor de memorie de adrese DS:ESI şi respectiv ES:EDI, urmată de incrementarea sau decrementarea registrelor SI şi DI. *Flag*-urile vor fi actualizate corespunzător pentru a reflecta rezultatul comparării.  Instrucţiunea **CMPSB** realizează compararea la nivel de octet, iar **CMPSW** la nivel de cuvânt, prima incrementând sau decrementând ESI şi EDI cu 1 iar ultima cu 2. În exemplul următor se compară două tablouri care conţin elemente reprezentate pe cuvânt pentru a se decide dacă primele 100 de elemente sunt sau nu identice: |  | Die **CMPS**-Anweisung hat die Aufgabe, Byte- oder Wortkettenvergleiche durchzuführen. Die Ausführung eines **CMPS**-Anweisung hat den Effekt, die Speicherstellen von DS:ESI- und ES:EDI-Adressen zu vergleichen, gefolgt vom Inkrementieren oder Dekrementieren der ESI- und EDI-Register. Die *Flags* werden entsprechend aktualisiert, um das Vergleichsergebnis widerzuspiegeln.  Die **CMPSB**-Anweisung führt den Vergleich auf Byte-Ebene und die **CMPSW** auf Wortebene durch, wobei ESI und EDI zuerst um 1 und EDI zuletzt um 2 erhöht oder erniedrigt werden. Im folgenden Beispiel werden zwei Tabellen, die durch das Wort dargestellte Elemente enthalten, verglichen, um zu entscheiden, ob die ersten 100 vorliegen von Elementen sind oder sind nicht identisch: |
| **mov** SI, OFFSET Tabloul  **mov** AX, SEG Tabloul  **mov** DS, AX  **mov** DI, OFFSET Tablou2  **mov** AX, SEG Tablou2  **mov** ES, AX  **mov** cx, 100  **cld**  **repe** **cmpsw**  **jne** TabDiferite  . . .  TabDiferite:  **dec** SI ; se actualizează regiştrii SI şi DI pentru  *; Die SI- und DI-Register werden aktualisiert, um auf das*  **dec** SI ; a puncta spre elementul prin care diferă  *; Element zu verweisen, um das sie sich unterscheiden*  **dec** DI  **dec** DI | | |
| 4.4 Execuţia repetată a unei instrucţiuni pe şiruri |  | 4.4 Wiederholte Ausführung eines String-Anweisung |
| Pentru execuţia repetată a unei instrucţiuni pe şiruri, limbajul de asamblare dispune de variante echivalente instrucţiunii de ciclare **LOOP**. Acestea sunt oferite de aşa numitele *prefixe de instrucţiune*. Sintaxa utilizării lor este: |  | Für die wiederholte Ausführung einer String-Anweisung verfügt die Assemblersprache über Varianten, die der **LOOP**-Zyklusanweisung entsprechen. Diese werden durch die sogenannten *Anweisungspräfixe* bereitgestellt. Die Syntax ihrer Verwendung lautet: |
| *prefix\_de\_instrucţiune instrucţiune\_pe\_şir*  (Anweisungspräfixe String-Anweisung) | | |
| În principiu este vorba despre un singur prefix de instrucţiune, şi anume **REP**. În cazul utilizării instrucţiunii **SCAS** sau **CMPS** apar două variante posibile de **REP**, Prefixul de instrucţiune **REP** impune execuţia repetată a instrucţiunii pe care o prefixează, până când valoarea din ECX devine 0. Dacă de la început avem ECX = 0, atunci instrucţiunea respectivă este inoperantă. Astfel, ciclul |  | Grundsätzlich gibt es nur ein Anweisungspräfix, nämlich **REP**. Bei Verwendung der **SCAS**- oder **CMPS**-Anweisung gibt es zwei mögliche **REP**-Varianten. Das **REP**-Anweisungspräfix erfordert die wiederholte Ausführung der Anweisung, bis der Wert in ECX 0 wird. Wenn von Anfang an ECX = 0 ist, ist die entsprechende Anweisung nicht funktionsfähig. So ist der Zyklus |
| Bucla: *instrucţiune\_pe\_şir (*String-Anweisung*)*  **loop** Bucla | | |
| este echivalent cu instrucţiunea |  | ist gleichbedeutend mit der Anweisung |
| **rep** *instrucţiune\_pe\_şir (*String-Anweisung*)* | | |
| În cazul utilizării prefixelor cu instrucţiunile **SCAS** sau **CMPS** condiţia verificată se completează, ţinându-se cont de valoarea *flag*-ului ZF. Acest lucru face ca prefixul **REP** să fie prezent în două variante.  Forma **REP** (echivalentă cu formele **REPE** – ***REP****eat while* ***E****qual* şi **REPZ** – ***REP****eat while* ***Z****ero*) provoacă execuţia repetată a instrucţiunilor **SCAS** sau **CMPS** până când CX devine 0 sau până când apare o nepotrivire (caz în care ZF va primi valoarea 0).  Asemănător, **REPNE** (***REP****eat while* ***N****ot* ***E****qual*, formă echivalentă cu **REPNZ** -***REP****eat if not* ***Z****ero*) provoacă execuţia repetată a instrucţiunii **SCAS** sau **CMPS** până când CX devine 0 sau până când apare o potrivire (caz în care ZF va primi valoarea 1).  Această deosebire există doar în cazul instrucţiunilor **SCAS** şi **CMPS** deoarece elesunt singurele care afectează vreun *flag* (şi anume ZF), permiţând rafinarea condiţiilor pe baza valorii *flag*-ului afectat. Rezultă deci, că în cazul instrucţiunilor **LODS**, **STOS** şi **MOVS** toate cele cinci mnemonici prezentate au acelaşi efect: repetarea cât timp CX <> 0.  Faţă de variantele cu **LOOP**, avantajul evident al folosirii acestor prefixe este comprimarea scrierii. |  | Bei Verwendung der Präfixe mit **SCAS**- oder **CMPS**-Anweisungen wird die überprüfte Bedingung unter Berücksichtigung des Werts der ZF-*Flagge* erfüllt. Dies bewirkt, dass das **REP**-Präfix in zwei Varianten vorhanden ist.  Das **REP**-Form (entspricht dem **REPE**- ***REP****eat while* ***E****qual* und **REPZ** – ***REP****eat while* ***Z****ero*) bewirkt die wiederholte Ausführung von **SCAS**- oder **CMPS**- Anweisungen, bis CX 0 wird oder eine Nichtübereinstimmung auftritt (in diesem Fall erhält ZF den Wert 0).  In ähnlicher Weise bewirkt REPNE (***REP****eat while* ***N****ot* ***E****qual*, eine Form, die **REPNZ** - ***REP****eat if not* ***Z****ero* entspricht) die wiederholte Ausführung der **SCAS**- oder **CMPS**-Anweisung, bis CX 0 wird oder bis eine Übereinstimmung auftritt (in diesem Fall erhält ZF den Wert 1).  Dieser Unterschied besteht nur im Fall von **SCAS**- und **CMPS**-Befehlen, da sie die einzigen sind, die ein *Flag* (dh ZF) beeinflussen, wodurch die Bedingungen basierend auf dem Wert des betroffenen *Flags* verfeinert werden können. Daraus folgt, dass für die Anweisungen **LODS**, **STOS** und **MOVS** alle fünf vorgestellten Mnemoniken den gleichen Effekt haben: Wiederholung solange CX <> 0.  Im Vergleich zu **LOOP**-Varianten liegt der offensichtliche Vorteil der Verwendung dieser Präfixe in der Schreibkomprimierung. |
|  |  |  |