

Curs 1 și 2

Contents

1. Sisteme de numerație și coduri.....	3
1. Nummerierungs- und Kodierungssysteme	3
1.1 Sisteme de numerație	3
1.1 Nummerierungssysteme	3
1.2 Conversia bazei de numerație	3
1.2 Umwandlung der Zahlenbasis	3
1.3 Coduri binare.....	6
1.3 Binäre Codes	6
1.3.1 Coduri ponderate	6
1.3.1 Gewichtete Codes.....	6
1.3.2 Coduri neponderate	7
1.3.2 Nicht gewichtete Codes.....	7
1.3.3 Bitul de paritate pentru detectia erorilor.....	8
1.3.3 Paritätsbit für Erkennen von Fehlern.....	8
2. Reprezentarea numerelor în calculator.....	9
2. Darstellung der Zahlen in den Computer	9
2.1 Întregi fără semn (<i>Unsigned Integers</i>).....	10
2.1 Vorzeichenlose ganze Zahlen (<i>Unsigned Integers</i>).....	10
2.2 Complementul lui 2 (<i>Two's Complement</i>)	11
2.2 2-Komplement Ganzzahl (<i>Two's Complement</i>)	11
2.3 Fracționare fără semn (<i>Unsigned fractional</i>)	14
2.3 Vorzeichenlose Bruchzahlen (<i>Unsigned fractional</i>)	14
2.4 Fracționare în complementul lui 2 (<i>Two's Complement signed fractional</i>).....	17
2.4 2-Komplement vorzeichen Bruchzahlen (<i>Two's Complement signed fractional</i>)	17
2.5 Codul Gray	17
2.5 Gray Code	17
2.6 Mărime și semn (<i>Signed magnitude</i>).....	19
2.6 Vorzeichenbehaftete Größenordnung (<i>Signed magnitude</i>)	19
2.7 Complementul lui 2 cu deplasament (<i>Offset two's Complement</i>)	20
2.7 Offset Zweierkomplement (<i>Offset two's Complement</i>).....	20
2.8 Complementul lui 1 (<i>One's Complement</i>)	20

2.8 Einerkomplement (<i>One's Complement</i>).....	20
2.9 Virgulă mobilă (<i>Floating point</i>)	21
2.9 Gleitkomma-Zahlen (<i>Floating point</i>).....	21
3. Reprezentări binare și ordini de plasare	25
3. Binäre Darstellungen und Platzierungsaufträge	25
3.1 Dimensiunea reprezentării.....	25
3.1 Darstellungsdimension	25
3.2 Organizarea și memorarea datelor.....	26
3.2 Daten organisieren und speichern	26
3.3 Tipuri elementare de date: dimensiuni ale standardelor de reprezentare	31
3.3 Elementare Datentypen: Dimensionen von die Darstellungsstandarden.....	31
3.4 Ordinea octeților într-o locație; mașini little-endian și mașini big-endian.....	32
3.4 Die Reihenfolge der Bytes an einem Speicherort; little-endian und big-endian Maschinen.....	32
3.5 Unități de capacitate a memoriei	37
3.5 Speicherkapazitätseinheiten	37
3.6 Codificarea caracterelor	37
3.6 Zeichenkodierung	37
3.7 Înmulțiri și împărțiri	40
3.7 Multiplikationen und Divisionen	40
3.8 Conversia la o locație de alte dimensiuni	43
3.8 An einen anderen Speicherort konvertieren	43

1. Sisteme de numerație și coduri

1.1 Sisteme de numerație

Sistemele numerice prelucrăză informația. În vederea prelucrării, informația trebuie să fie *codificată*. Pentru codificare, se utilizează un anumit tip de reprezentare.

Sistemul de numerație este format din totalitatea regulilor de reprezentare a numerelor cu ajutorul unor simboluri numite cifre. Sistemele de numerație pot fi *poziționale* (valoarea unei cifre este determinată de poziția sa în cadrul numărului) sau *nepozitionale*.

Un număr „ N “ într-un sistem pozitional poate fi reprezentat într-o bază de numerație „ b “ astfel:

$$N = a_{q-1}b^{q-1} + \dots + a_0b^0 + \dots + a_{-p}b^{-p} = \sum_{i=-p}^{q-1} a_i b^i$$

unde baza „ b “ este un număr întreg mai mare ca 1 și a_i sunt întregi în gama $0 \leq a_i \leq b - 1$. Numărul „ N “ în baza „ b “ se notează astfel: $(N)_b$. Atunci când baza nu este specificată, ea este implicit 10 (deoarece sistemul zecimal este cel mai utilizat în practică).

Când baza $b = 2$, reprezentarea numerică se numește *sistem numeric binar*. Complementul unei cifre „ a “, notat cu „ \bar{a} “, în baza „ b “ este definit ca $\bar{a} = (b - 1) - a$. În sistemul numeric binar $\bar{0} = 1$ și $\bar{1} = 0$.

1.2 Conversia bazei de numerație

În multe aplicații practice se pune problema conversiei unui număr exprimat în baza „ b_1 “ în altă bază „ b_2 “. În procesul de conversie distingem două cazuri:

1. Nummerierungs- und Kodierungssysteme

1.1 Nummerierungssysteme

Numerische Systeme verarbeiten die Informationen. Zur Verarbeitung müssen die Informationen *codiert* werden. Zur Kodierung wird eine bestimmte Art der Darstellung verwendet.

Das *Nummerierungssystem* besteht aus allen Regeln für die Darstellung von Zahlen durch Symbole, die Ziffern genannt werden. Numerische Systeme können *positionell* sein (der Wert einer Ziffer wird durch ihre Position innerhalb der Zahl bestimmt) oder *nicht positionell* sein.

Eine „ N “ – Nummer in einem Positionssystem kann in einer Zahlobasis „ b “ wie folgt dargestellt werden:

wobei Basis „ b “ eine ganze Zahl größer als 1 ist und a_i eine ganze Zahl im Bereich $0 \leq a_i \leq b - 1$ ist. Die Zahl „ N “ in der Basis „ b “ lautet wie folgt: $(N)_b$. Wenn keine Basis angegeben wird, ist der Standard 10 (weil das Dezimalsystem in der Praxis am häufigsten verwendet wird).

Wenn die Basis $b = 2$ ist, wird die numerische Darstellung als binäres numerisches System bezeichnet.

Das Komplement einer Ziffer „ a “, gekennzeichnet mit „ \bar{a} “, ist in der Basis „ b “ als $\bar{a} = (b - 1) - a$ definiert. Im binären Zahlsystem $\bar{0} = 1$ und $\bar{1} = 0$.

1.2 Umwandlung der Zahlobasis

In vielen praktischen Anwendungen besteht das Problem die Umwandlung einer Zahl, die in der Basis "b₁" ausgedrückt wird, in eine andere Basis "b₂" zu erfassen. Bei der Umwandlung unterscheiden wir zwei Fälle:

- a) $b_1 < b_2$
b) $b_1 > b_2$

În cazul a) conversia implică exprimarea numărului $(N)b_1$ ca un polinom în puterile lui „ b_1 “ și evaluarea polinomului folosind aritmetică în baza „ b_2 “.

Exemplu: Pentru $b_1 = 3$, $b_2 = 10$ și $(N)_3 = 2201.1$ vom avea:

$$(N)_{10} = 2 \cdot 3^3 + 2 \cdot 3^2 + 0 \cdot 3^1 + 1 \cdot 3^0 + 1 \cdot 3^{-1} \\ = 54 + 18 + 0 + 1 + 0,3 = 73,3$$

În cazul b) este mai convenabil să utilizăm aritmetică în baza „ b_1 “. Conversia numărului se face prin conversia separată a părții întregi și a părții fractionare a acestuia. Pentru conversia părții întregi a numărului, aceasta se împarte la baza „ b_2 “ obținându-se astfel un cât și un rest; se reține restul și se continuă cu împărțirea la baza „ b_2 “ a câtului. Algoritmul se oprește în momentul în care avem câtul 0. Partea întreagă a rezultatului se obține prin scrierea resturilor în ordinea inversă a generării lor.

Pentru conversia părții fractionare a numărului, aceasta se înmulțește cu baza „ b_2 “, obținându-se astfel un număr format dintr-o parte întreagă și o parte fractionară; se reține partea întreagă și se continuă cu înmulțirea cu baza „ b_2 “ a părții fractionare obținute. Procesul continuă până la obținerea preciziei dorite.

Exemplu:

Pentru $b_1 = 10$, $b_2 = 4$ și $(N)_{10} = 47.4$ vom avea:

1. Conversia părții întregi:

$$47:4 = 11 \text{ rest } 3 \\ 11:4 = 2 \text{ rest } 3 \\ 2:4 = 0 \text{ rest } 2$$

Partea întreagă a rezultatului este $(233)_4$.

2. Conversia părții fractionare:

$$0.4 \cdot 4 = 1.6 \Rightarrow 1$$

- a) $b_1 < b_2$
b) $b_1 > b_2$

Im Falle von 1) die Umwandlung beinhaltet zwei Schritte: der Zahl $(N)b_1$ als Polynom in Potenzen von "b₁" und die Auswertung des Polynoms unter Verwendung der auf "b₂" basierenden Arithmetik zu äußern.

Beispiel: Für $b_1 = 3$, $b_2 = 10$ und $(N)_3 = 2201.1$ haben wir:

$$(N)_{10} = 2 \cdot 3^3 + 2 \cdot 3^2 + 0 \cdot 3^1 + 1 \cdot 3^0 + 1 \cdot 3^{-1} \\ = 54 + 18 + 0 + 1 + 0,3 = 73,3$$

Im Fall 2) ist es günstiger, die Arithmetik in der Basis "b₁" zu verwenden. Die Konvertierung der Nummer erfolgt durch separate Konvertierung des Ganzteils und des Bruchteils. Für die Umwandlung des Ganzteils der Zahl wird diese in die Basis "b₂" unterteilt, wodurch sowohl ein Quotient als auch ein Rest erhalten wird; der Rest wird beibehalten und mit der Division an der Basis "b₂" des Quotienten fortgesetzt. Der Algorithmus stoppt, wenn der Quotient 0 ist. Der Ganzteil des Ergebnisses wird durch Schreiben der Reste in umgekehrter Reihenfolge ihrer Erzeugung erhalten.

Um den gebrochenen Teil der Zahl zu konvertieren, multiplizieren Sie ihn mit der Basis "b₂", um eine Zahl zu erhalten, die aus einem Ganzen und einem gebrochenen Teil besteht. Behalten Sie der Ganzteil und fahren Sie mit der Multiplikation des gebrochenen Teils mit "b₂" fort. Der Prozess wird fortgesetzt, bis die gewünschte Genauigkeit erreicht ist.

Beispiel:

Für $b_1 = 10$, $b_2 = 4$ und $(N)_{10} = 47.4$ haben wir:

1. Die Umwandlung des Ganzteils:

$$47:4 = 11 \text{ Rest } 3 \\ 11:4 = 2 \text{ Rest } 3 \\ 2:4 = 0 \text{ Rest } 2$$

Der Ganzteil des Ergebnisses ist: $(233)_4$.

2. Die Umwandlung des Bruchteils:

$$0.4 \cdot 4 = 1.6 \Rightarrow 1$$

$$0.6 \cdot 4 = 2.4 \Rightarrow 2$$

Partea fracționară a rezultatului este
 $(1212\dots)_4$.

În concluzie $(N)_4 = 223.1212\dots$

Alte Exemple

1. Pentru $b_1 = 10$, $b_2 = 6$ și $(N)_{10} = 985437$ avem:

$$\begin{aligned} 985437 : 6 &= 164239 \text{ rest } 3 \\ 164239 : 6 &= 27373 \text{ rest } 1 \\ 27373 : 6 &= 4562 \text{ rest } 1 \\ 4562 : 6 &= 760 \text{ rest } 2 \\ 760 : 6 &= 126 \text{ rest } 4 \\ 126 : 6 &= 21 \text{ rest } 0 \\ 21 : 6 &= 3 \text{ rest } 3 \\ 3 : 6 &= 0 \text{ rest } 3 \end{aligned}$$

Deci avem $(985437)_{10} = (33042113)_6$.

2. Pentru $b_1 = 10$, $b_2 = 16$ și $(N)_{10} = 985437$ avem:

$$\begin{aligned} 985437 : 16 &= 61589 \text{ rest } 13 \text{ (D)} \\ 61589 : 16 &= 3849 \text{ rest } 5 \\ 3849 : 16 &= 240 \text{ rest } 9 \\ 240 : 16 &= 15 \text{ rest } 0 \\ 15 : 16 &= 0 \text{ rest } 15 \text{ (F)} \end{aligned}$$

Deci avem $(985437)_{10} = (\text{F095D})_{16}$.

Un caz aparte îl constituie conversia numerelor octale și hexazecimale în binar și invers. Atunci se poate folosi o procedură de conversie mult mai simplă. Pentru aceasta fiecare cifră octală sau hexazecimală se exprimă prin 3, respectiv 4 cifre binare.

Exemplu:

$$\begin{aligned} (123.4)_8 &= (001\ 010\ 011.100) = \\ &(1010011.1)_2 \\ (\text{B7.2})_{16} &= (1011\ 0111.0010) = \\ &(10110111.001)_2 \end{aligned}$$

La conversia din binar în octal sau hexazecimal se fac grupări de câte 3, respectiv 4 cifre binare.

Exemplu:

$$(1010110.0101)_2 = (001\ 010\ 110.010\ 100)$$

$$0.6 \cdot 4 = 2.4 \Rightarrow 2$$

Der Bruchteil des Ergebnisses ist:
 $(1212\dots)_4$.

Abschließend, $(N)_4 = 223.1212\dots$

Andere Beispiele

1. Für $b_1 = 10$, $b_2 = 6$ und $(N)_{10} = 985437$ haben wir:

$$\begin{aligned} 985437 : 6 &= 164239 \text{ Rest } 3 \\ 164239 : 6 &= 27373 \text{ Rest } 1 \\ 27373 : 6 &= 4562 \text{ Rest } 1 \\ 4562 : 6 &= 760 \text{ Rest } 2 \\ 760 : 6 &= 126 \text{ Rest } 4 \\ 126 : 6 &= 21 \text{ Rest } 0 \\ 21 : 6 &= 3 \text{ Rest } 3 \\ 3 : 6 &= 0 \text{ Rest } 3 \end{aligned}$$

Abschließend $(985437)_{10} = (33042113)_6$.

2. Für $b_1 = 10$, $b_2 = 16$ und $(N)_{10} = 985437$ haben wir:

$$\begin{aligned} 985437 : 16 &= 61589 \text{ Rest } 13 \text{ (D)} \\ 61589 : 16 &= 3849 \text{ Rest } 5 \\ 3849 : 16 &= 240 \text{ Rest } 9 \\ 240 : 16 &= 15 \text{ Rest } 0 \\ 15 : 16 &= 0 \text{ Rest } 15 \text{ (F)} \end{aligned}$$

Abschließend $(985437)_{10} = (\text{F095D})_{16}$.

Ein Sonderfall ist die Umwandlung von Oktal- und Hexadezimalzahlen in Binärzahlen und umgekehrt. Dann kann ein wesentlich einfacherer Umwandlungsverfahren verwendet werden. Zu diesem Zweck wird jede Oktal- oder Hexadezimalziffer durch 3 bzw. 4 Binärziffern ausgedrückt.

Beispiel:

$$\begin{aligned} (123.4)_8 &= (001\ 010\ 011.100) = \\ &(1010011.1)_2 \\ (\text{B7.2})_{16} &= (1011\ 0111.0010) = \\ &(10110111.001)_2 \end{aligned}$$

Bei der Umwandlung von Binär- in Oktal- und Hexadezimalzahlen werden Gruppen von 3 bzw. 4 Binärziffern gebildet.

Beispiel:

$$(1010110.0101)_2 = (001\ 010\ 110.010\ 100) =$$

$$= (126.24)_8$$

$$(1011110.011)_2 = (0101 \quad 1110.0110) =$$

$$(5E.6)_{16}$$

1.3 Coduri binare

Deși sistemul de numerație binar are multe avantaje practice și este foarte utilizat în calculatoarele numerice, în multe cazuri este convenabil să lucrăm cu sistemul zecimal, în special acolo unde comunicația între om și mașină este intensă. Pentru a simplifica problema comunicației au fost definite un număr de coduri astfel încât cifrele zecimale să fie reprezentate prin succesiuni de cifre binare.

Pentru a reprezenta cele 10 cifre zecimale este suficient să folosim 4 cifre binare. Codurile binare se pot împărti în două clase: *ponderate și neponderate*.

1.3.1 Coduri ponderate

Caracteristica principală a codurilor ponderate este aceea că fiecare cifre binare îi este asociată o „pondere“. Pentru fiecare grup de 4 biți suma ponderilor acelor cifre binare a căror valoare este 1 este egală cu cifra zecimală pe care o reprezintă. O cifră zecimală într-un cod ponderat se scrie astfel:

$$N = \sum_{i=0}^3 a_i b_i$$

unde $a_i = 0$ sau 1 .

În Tabelul 1 se dau trei exemple de coduri binare ponderate. Primul cod se numește **BCD** (Binary Coded Decimal), deoarece pentru obținerea codului fiecare cifră zecimală este convertită în binar.

$$(126.24)_8$$

$$(1011110.011)_2 = (0101 \quad 1110.0110) =$$

$$(5E.6)_{16}$$

1.3 Binäre Codes

Obwohl das binäre Nummerierungssystem viele praktische Vorteile hat und in numerischen Computern weit verbreitet ist, ist es oft zweckmäßig, mit dem Dezimalsystem zu arbeiten, insbesondere wenn der Mensch-Maschine-Informationsaustausch intensiv ist. Um das Kommunikationsproblem zu vereinfachen, wurde eine Anzahl von Codes definiert, so dass die Dezimalzahlen durch binäre Ziffernfolgen dargestellt werden. Zur Darstellung der 10 Dezimalstellen verwenden Sie einfach 4 Binärstellen. Binäre Codes können in zwei Klassen unterteilt werden: gewichtet und nicht übereinstimmend.

1.3.1 Gewichtete Codes

Das Hauptmerkmal gewichteter Codes besteht darin, dass jeder Binärzahl eine "Gewichtung" zugeordnet ist. Für jede 4-Bit-Gruppe entspricht die Summe dieser Binärziffern, deren Wert 1 ist, der Dezimalzahl, die sie darstellen. Eine Dezimalzahl in einem gewichteten Code wird wie folgt geschrieben:

wobei $a_i = 0$ oder 1 .

Tabelul 1 conține trei exemple pentru coduri binare ponderate. Primul cod se numește **BCD** (Binary Coded Decimal), deoarece pentru obținerea codului fiecare cifră zecimală este convertită în binar.

Tabelul 1. Trei exemple de coduri ponderate (Drei Beispiele für gewichtete Codes)

Cifră Zecimală (Dezimalzahl)	b ₃ b ₂ b ₁ b ₀				b ₃ b ₂ b ₁ b ₀				pondere negativă (negatives Gewicht)			
	8	4	2	1	2	4	2	1	6	4	2	-3
0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1	0	1	0	1
2	0	0	1	0	0	0	1	0	0	0	1	0
3	0	0	1	1	0	0	1	1	1	0	0	1
4	0	1	0	0	0	1	0	0	0	1	0	0
5	0	1	0	1	1	0	1	1	1	0	1	1
6	0	1	1	0	1	1	0	0	0	1	1	0
7	0	1	1	1	1	1	0	1	1	1	0	1
8	1	0	0	0	1	1	1	0	1	0	1	0
9	1	0	0	1	1	1	1	1	1	1	1	1

1.3.2 Coduri neponderate

Cele mai utilizate coduri neponderate sunt codul *Exces 3* și codul *Gray*. Codul Exces 3 este format prin adăugarea lui 0011 la fiecare cuvânt de cod din BCD. Este un cod autocomplementar și posedă anumite proprietăți care l-au făcut practic. Din acest cod s-a eliminat combinația 0000, care ar putea fi confundată cu lipsa de informație.

În multe aplicații practice, analog conversiei digitale este de dorit a se folosi codurile în care toate cuvintele de cod succesive diferă doar printr-o cifră. Codurile care au o astfel de proprietate se numesc *coduri ciclice*.

Un astfel de cod este codul Gray. Codul Gray de „*n*“ biți face parte din clasa *codurilor reflectate*. Termenul „reflectate“ este folosit pentru a desemna coduri care au următoarea proprietate: cuvântul de cod de „*n*“ biți poate fi generat prin reflectarea codului de „*n*-1“ biți.

1.3.2 Nicht gewichtete Codes

Die am meisten verwendete Codes sind Code Excess 3 und Gray Code. Code *Excess 3* wird durch Hinzufügen von 0011 zu jedem BCD-Codewort gebildet. Es ist ein selbstkomplementären Code und besitzt bestimmte Eigenschaften, die ihn praktisch machen. Aus diesem Code ist die 0000-Kombination gelöscht, die mit dem Mangel an Informationen verwechselt werden könnte.

In vielen praktischen Anwendungen, bei der digitalen Umwandlung, ist es wünschenswert, Codes zu verwenden, bei denen sich alle aufeinanderfolgenden Codewörter nur um eine Ziffer unterscheiden. Codes, die über eine solche Eigenschaft verfügen, werden als *zyklische Codes* bezeichnet.

Ein solcher Code ist der Gray-Code. Gray 'n'-Bitcode ist Teil der reflektierten Codeklasse. Der Begriff "reflektiert" wird verwendet, um Codes zu bezeichnen, die die folgende Eigenschaft haben: Das Codewort von "*n*" Bits kann durch Reflektion der "*n*-1" Bits Code erzeugt werden.

Tabelul 2. Obținerea codului Gray de 3 biți din cel de 2 biți (Herstellung des 3-Bit-Gray-Codes

von den 2-Bit-Code)

Gray 2 biți (2 Bits)		Gray 3 biți (3 Bits)		
0	0	0	0	0
0	1	0	0	1
1	1	0	1	1
1	0	0	1	0
		1	1	0
		1	1	1
		1	0	1
		1	0	0

În **Tabelul 3** sunt prezentate cifrele zecimale în cod Exces 3, respectiv Gray:

Tabelul 3. Codificarea cifrelor zecimale în codurile Exces 3 și Gray (Die Dezimalstellen in Überschuss 3 (Exces 3) oder Gray-Code)

cifră zecimală (Dezimalzahl)	Exces 3	Gray
0	0 0 1 1	0 0 0 0
1	0 1 0 0	0 0 0 1
2	0 1 0 1	0 0 1 1
3	0 1 1 0	0 0 1 0
4	0 1 1 1	0 1 1 0
5	1 0 0 0	0 1 1 1
6	1 0 0 1	0 1 0 1
7	1 0 1 0	0 1 0 0
8	1 0 1 1	1 1 0 0
9	1 1 0 0	1 1 0 1

1.3.3 Bitul de paritate pentru detecția erorilor

În procesul de transmitere a informației în sistemele numerice, aceasta poate fi alterată. În vederea determinării corectitudinii informației receptionate, se pot folosi coduri detectoare și corectoare de erori.

Codurile detectoare de erori au următoarea proprietate: *apariția unei singure erori transformă un cuvânt valid într-un cuvânt invalid.*

O metodă pentru detecția erorilor este *metoda bitului de paritate*. Ideea de bază în controlul de paritate este de a adăuga o cifră binară în plus la fiecare cod cuvânt al unui

Tabelul 3 zeigt die Dezimalstellen in Überschuss 3 (Exces 3) oder Gray-Code.

1.3.3 Paritätsbit für Erkennen von Fehlern

Bei der Übertragung von Informationen in numerischen Systemen kann diese geändert werden. Um die Genauigkeit der empfangenen Informationen zu bestimmen, können Fehlererkennungs- und Korrekturcodes verwendet werden.

Fehlererkennung Codes haben die folgende Eigenschaft: *Das Auftreten eines einzelnen Fehlers wandelt ein gültiges Wort in ein ungültiges Wort um.*

Eine Methode zum Erkennen von Fehlern ist die *Paritätsbit Methode*. Die Grundidee bei der Paritätskontrolle besteht darin, jedem Wortcode eines gegebenen Codes eine

cod dat, pentru a face ca numărul de biți de 1 din fiecare cuvânt să fie impar sau par.

zusätzliche Binärziffer hinzuzufügen, um die Anzahl der Bits von 1 in jedem Wort ungerade oder gerade zu machen.

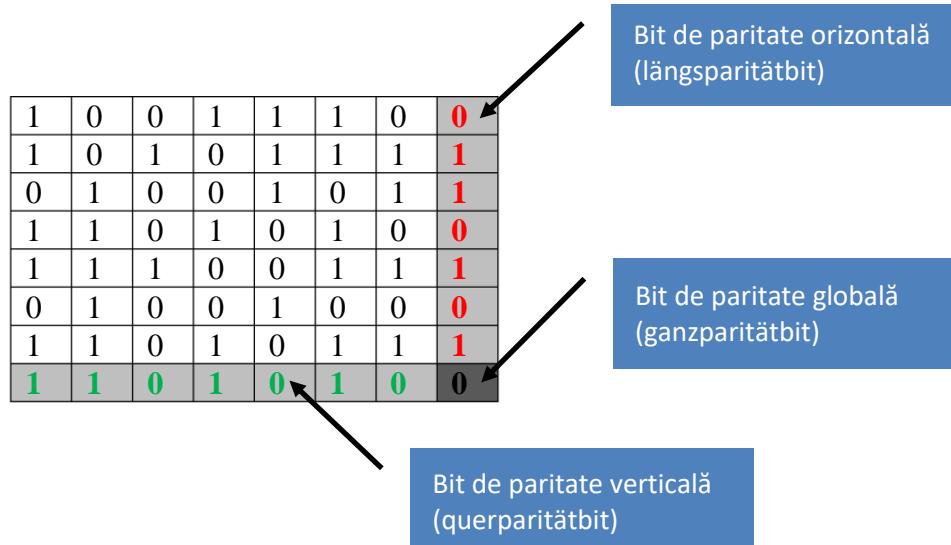


Figura 1. Bitul de paritate (Das Paritätbit)

2. Reprezentarea numerelor în calculator

Sistemele de numerație binare sunt utilizate în aproape toate sistemele numerice, inclusiv în aplicațiile de procesare a semnalelor digitale (DSP), în rețele și în sistemele de calcul. Înainte de a alege un sistem de numerație, este important să-i înțelegem avantajele și dezavantajele și, de asemenea, să știm cum să convertim numerele dintr-un sistem în altul.

În continuare vom descrie următoarele sisteme de numerotare, avantajele și dezavantajele fiecărui sistem și tranziția între diferite sisteme.

- Întregi fără semn
- Complementul față de 2
- Fracționare fără semn
- Fracționare cu semn în Complementul față de 2
- Cod Gray
- Mărime și semn

2. Darstellung der Zahlen in den Computer

Binäre Zahlensysteme werden in nahezu allen digitalen Systemen, einschließlich der digitalen Signalverarbeitung (DSP), Netzwerk- und Computer verwendet.

Bevor Sie sich für ein Nummerierungssystem entscheiden, müssen Sie die Vor- und Nachteile der einzelnen Systeme verstehen.

Nachfolgend werden die folgenden Zahlensysteme beschrieben, die Vor- und Nachteile der einzelnen Systeme sowie die Umstellung zwischen verschiedenen Systemen.

- vorzeichenlose Ganzzahl
- 2-Komplement Ganzzahl
- Vorzeichenlose Bruchzahlen
- 2-Komplement vorzeichen Bruchzahlen
- Gray-Code
- vorzeichenbehaftete Größenordnung

- Complementul față de 2 deplasat
- Complementul față de 1
- Virgulă mobilă

- Offset von 2-Komplement
- Einerkomplement
- Fließkomma

2.1 Întregi fără semn (*Unsigned Integers*)

Cel mai bine cunoscut sistem de numerație este cel numit „Întregi fără semn“. La fel ca în sistemul zecimal, în Întregi fără semn se folosește o regulă simplă, de natură binară, de asociere a valorilor cifrelor. **Poziția unei cifre binare determină valoarea sa (adică valoarea asociată poziției unei cifre binare este $2^{Pozitie}$).** Această metodă de reprezentare este exact ca sistemul zecimal, unde valoarea asociată poziției unei cifre zecimale este $10^{Pozitie}$.

2.1 Vorzeichenlose ganze Zahlen (*Unsigned Integers*)

Das bekannteste Nummerierungssystem ist die vorzeichenlose Ganzzahldarstellung. Wie das Dezimalsystem verwenden vorzeichenlose Ganzzahlen einen einfachen binären Platzwert. **Die Position einer Ziffer bestimmt ihren Wert (d. H. Der Platzwert einer Ziffer ist $2^{Position}$).** Diese Darstellung entspricht genau dem Dezimalsystem, bei dem der Platzwert $10^{Position}$ ist.

Ponderea (valoarea asociată poziției) - (Platzwert)	2^4	2^3	2^2	2^1	2^0
Poziția (Position)	4	3	2	1	0
Numărul binar (Binäre Zahl)	1	0	0	1	1

Figura 2. Poziția bitilor în sistemul de numerație Întregi fără semn (Bit Position in dem Vorzeichenlose ganze Nummerierungssystem)

Tabelul 4. Conversia numerelor Întregi fără semn (Die Umwandlung von Ganzzahlen ohne Vorzeichen)

Întregi fără semn (Ganzzahlen ohne Vorzeichen)	Valoarea zecimală (Dezimalwert)	Conversie (Umwandlung)
01000	8	$0+2^3+0+0+0=8$
10011	19	$2^4+0+0+2^1+2^0=16+2+1=19$
11011	27	$2^4+2^3+0+2^1+2^0=16+8+2+1=27$

Se pot realiza cu ușurință operații aritmetice pe Întregi fără semn, respectând aceleași reguli ca în cazul numerelor zecimale. Dar, pentru numerele binare, cifra este transportată (*carry*) după 1 și nu după 9 (adică atunci când se adună două cifre 1, se va scrie un 0 pe poziția corespunzătoare și un 1 se va transportat către poziția următoare). Figura 3 prezintă modul de realizare a adunării a două numere Întregi fără semn.

$$\begin{array}{r} 11011 \\ + 10011 \\ \hline 1\ 01110 \end{array} = \begin{array}{r} 27 \\ + 19 \\ \hline 46 \end{array}$$

Figura 3. Adunarea numerelor Întregi fără semn (Addierte Ganzzahl ohne Vorzeichen)

Sistemul de numerație Întregi fără semn este foarte folosit. Principala limitare a acestui sistem de numerație constă în aceea că nu se pot reprezenta decât numerele cuprinse între 0 și (2^N-1) . Majoritatea sistemelor de prelucrare a semnalelor digitale au nevoie să stocheze atât numere pozitive, cât și numere negative.

2.2 Complementul lui 2 (Two's Complement)

Cel mai folosit sistem de numerație care permite reprezentarea atât a numerelor pozitive, cât și a numerelor negative, este *Complementul lui 2*. Acest sistem de numerație este similar sistemului de numerație Întregi fără semn, cu excepția faptului că semnul bitului cel mai semnificativ (MSB) este negat. Pentru un număr pe N biți, bitul 0 are tot ponderea 2^0 , bitul 1 are tot ponderea 2^1 , bitul $N-2$ are tot ponderea 2^{N-2} , dar bitul $N-1$ (adică MSB-ul) are ponderea $-2^{(N-1)}$. Tabelul 5 prezintă valoarea zecimală corespunzătoare fiecărei poziții în cazul unei reprezentări în Complementul lui 2 pe 5 biți.

Sie können arithmetische Operationen mit vorzeichenlosen Zahlen problemlos durchführen, indem Sie dieselben Regeln wie für Dezimalzahlen verwenden. Bei Binärzahlen wird die Ziffer jedoch nach 1 statt nach 9 übertragen (d. H., Wenn zwei 1en addiert werden, wird eine 0 an die entsprechende Position gesetzt und eine 1 wird zur nächsten Position befördert). Figura 3 zeigt, wie zwei vorzeichenlose Ganzzahlen zusammengefügt werden.

$$= \begin{array}{r} 27 \\ + 19 \\ \hline 46 \end{array}$$

Das vorzeichenlose Integer-Nummerierungssystem ist weit verbreitet. Die Hauptbeschränkung dieses Zahlensystems besteht darin, dass es nur die Zahlen im Bereich von 0 bis (2^N-1) speichern kann. Die meisten Signalverarbeitungssysteme müssen sowohl positive als auch negative Zahlen speichern.

2.2 2-Komplement Ganzzahl (Two's Complement)

Das am häufigsten verwendete Nummerierungssystem, das sowohl positive als auch negative Zahlen speichern kann, ist die Zweierkomplement-Ganzzahl. Dieses System ist einer vorzeichenlosen Ganzzahl ähnlich, es sei denn, das Vorzeichen des höchswertigen Bits (MSB) ist negiert. Für eine N -Bit-Nummer hat beispielsweise Bit 0 einen Wert von 2^0 , Bit 1 hat einen Wert von 2^1 , Bit $N-2$ hat einen Wert von 2^{N-2} und Bit $N-1$ (d. H. das MSB) hat einen Wert von $-2^{(N-1)}$. **Tabelul 5** zeigt den Dezimalwert für jede Position in einer 5-Bit-Zweierkomplement-Ganzzahl.

Tabelul 5. Valorile întregi ale ponderilor biților în cazul reprezentării în Complementul lui 2 a unui număr pe 5 biți (Zweierkomplement Integer-Werte für eine 5-Bit-Zahl)

Pozitie (Position)	Ponderea pozitiei (Stellenwert)	Valoarea zecimala (Dezimalwert)
0 (LSB)	2^0	1
1	2^1	2
2	2^2	4
3	2^3	8
4 (MSB)	-2^4	-16

Câteva exemple:

Einige Beispiele:

$$01000 = -2^4 \times 0 + 2^3 \times 1 + 2^2 \times 0 + 2^1 \times 0 + 2^0 \times 0 = +8$$

$$11000 = -2^4 \times 1 + 2^3 \times 1 + 2^2 \times 0 + 2^1 \times 0 + 2^0 \times 0 = -8$$

$$10000 = -2^4 \times 1 + 2^3 \times 0 + 2^2 \times 0 + 2^1 \times 0 + 2^0 \times 0 = -16$$

$$10111 = -2^4 \times 1 + 2^3 \times 0 + 2^2 \times 1 + 2^1 \times 1 + 2^0 \times 1 = -9$$

Complementul lui 2 permite reprezentarea numerelor cuprinse între $-2^{(N-1)}$ și $2^{(N-1)} - 1$. Pentru a obține complementul unui număr în acest sistem de numerație, se inversează toți biții și se adaugă 1. Următorii pași ilustrează cu titlu de exemplu modul în care se obține complementul numărului 9.

1. Se reprezintă numărul în Complementul lui 2: $9 = 01001$

2. Se inversează toți biții: 10110

3. Se adună 1: $(10110 + 1) = 10111$

Cel mai mare avantaj al sistemelor de numerație bazate pe Complementul lui 2 constă în aceea că adunarea și scăderea este identică cu adunarea și scăderea numerelor întregi fără semn. Trebuie însă efectuată extensia de semn înainte de realizarea operației, iar transportul de la nivelul bitului cel mai semnificativ trebuie ignorat. Figura 4 prezintă câteva cazuri de adunare a numerelor reprezentate în Complementul lui 2.

Zweierkomplement-Ganzzahlen

repräsentieren Zahlen im Bereich von $-2^{(N-1)}$ bis $2^{(N-1)} - 1$. Um eine Zweierkomplement-Ganzzahl zu negieren, invertieren Sie einfach die Bits und fügen 1 hinzu. Die folgenden Schritte zeigen beispielsweise, wie die Zahl 9 zu -9 negiert wird.

1. Ersetzen Sie die binären Werte der Dezimalzahl: $9 = 01001$

2. Invertieren Sie die Bits: 10110

3. Hinzufügen: $(10110+1) = 10111$

Der größte Vorteil des Zweierkomplement-Zahlensystems besteht darin, dass das Addieren und Subtrahieren von Zweierkomplement Nummern dem Hinzufügen und Subtrahieren von vorzeichenlosen Zahlen entspricht. Die Zeichen Erweiterung muss jedoch vor der Operation ausgeführt werden, und die Ausführung des Addierers sollte ignoriert werden. Figur 4 zeigt, wie zwei 2-Bit-Zweierkomplementzahlen hinzugefügt werden.

Pozitiv + pozitiv

1 001

Negativ + pozitiv

-1 111

Negativ + negativ

-1 111

$$\begin{array}{r} + 1 \\ \hline 2 \end{array} \quad \begin{array}{r} + 001 \\ \hline 010 \end{array} \quad \begin{array}{r} + 1 \\ \hline 0 \end{array} \quad \begin{array}{r} + 001 \\ \hline 1 000 \end{array} \quad \begin{array}{r} + -1 \\ \hline -2 \end{array} \quad \begin{array}{r} + 111 \\ \hline 1 110 \end{array}$$

Figura 4. Adunarea numerelor în Complementul lui 2 (Zweierkomplement-Integeraddition)

Observații

1. Bitul de transport de la nivelul MSB se ignoră.
2. Toate numerele pozitive au MSB = 0, iar toate numerele negative au MSB = 1. De aceea, MSB indică semnul.
3. Dacă cele două numere care trebuie adunate nu sunt reprezentate pe același număr de biți, operația de adunare se va face pe lățimea celui mai mare dintre ele. La nivelul numărului mai mic este atunci necesar să se realizeze operațiunea de *extensie de semn*, care constă în replicarea MSB-ului până când se atinge o lățime egală cu cea a celuilalt operand. *Extensia de semn* nu modifică valoarea numărului!

Bemerkungen

1. Transportbit auf MSB-Ebene wird ignoriert.
2. Alle positiven Zahlen haben MSB = 0 und alle negativen Zahlen haben MSB = 1. Daher gibt der MSB das Vorzeichen an.
3. Wenn die zwei zu sammelnden Zahlen nicht mit der gleichen Anzahl von Bits dargestellt sind, wird der Zusammensetzungsvorgang mit der breitesten von ihnen ausgeführt. Bei der niedrigeren Zahl muss dann die *Vorzeichenerweiterungsoperation* ausgeführt werden, die darin besteht, das MSB zu replizieren, bis eine Breite erreicht ist, die der des anderen Operanden entspricht. Die *Vorzeichenerweiterung* ändert den Wert der Zahl nicht!

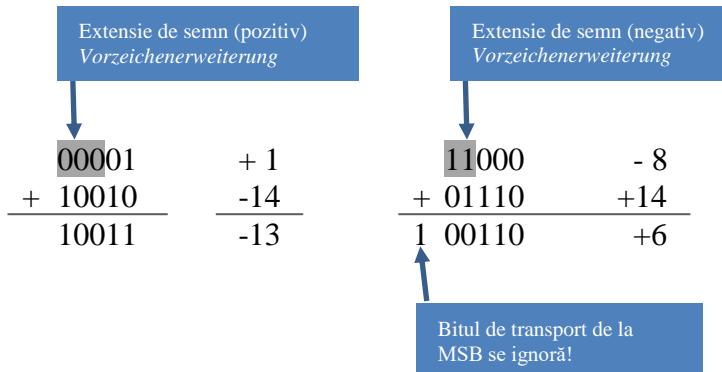


Figura 5. Extensia de semn pentru adunarea numerelor în Complementul lui 2 (Vorzeichenerweiterung für Zweierkomplement-Integeraddition)

4. Dacă la Întregi cu semn nu se punea problema depășirii (*overflow*), la Complementul lui 2 există posibilitatea ca rezultatul adunării să fie un număr în afara intervalului de reprezentare. Depășirea se detectează astfel: dacă biții de transport de la nivelul bitului MSB și al bitului alăturat sunt diferiți, atunci avem depășire.

Wenn der Überlauf mit dem Vorzeichen nicht aufgetreten ist, besteht in 2-Komplement Ganzzahl die Möglichkeit, dass das Ergebnis der Assembly eine Zahl außerhalb des Darstellungsbereichs ist. Das Überholen wird wie folgt erkannt: Wenn die Transportbits vom MSB-Bit und den benachbarten Bits verschieden sind, haben wir überschritten.

Biți de transport diferiți : OVERFLOW!

Biți de transport identici : NU APARE (kein) OVERFLOW

Biți de transport identici : NU APARE (kein) OVERFLOW

Figura 6. Depășirea la adunarea numerelor în Complementul lui 2 (Überlauf für Zweierkomplement-Integeraddition)

2.3 Fracționare fără semn (*Unsigned fractional*)

În DSP (*Digital Signal Processing*) și alte sisteme, este adeseori necesar să se stocheze numere care au atât o parte întreagă, cât și o parte fracțională. Întrucât anumite poziții binare pot fi negative, acest sistem de numerație poate stoca atât numere supraunitare cât și numere subunitare. Ponderea unei cifre în Fracționare fără semn este $2^{Position}$, unde poziția poate fi atât pozitivă, cât și negativă (vezi Figura 7). De aceea, sistemul de numerație Fracționare fără semn este un superset al sistemului de numerație Întregi fără semn.

Tabelul 6. Ponderile bițiilor în cazul reprezentării în Fracționare fără semn a unui număr pe 5 biți (Vorzeichenlose gebrochene Werte)**Tabelul 6** prezintă ponderile bițiilor în cazul reprezentării în sistemul de numerație Fracționare fără semn a unui număr pe 5 biți, iar **Tabelul 7** prezintă câteva exemple.

Ponderea (valoarea asociată poziției) - (Platzwert)	2^2	2^1	2^0	2^{-1}	2^{-2}
Poziția (Position)	2	1	0	-1	-2
Numărul binar (Binäre Zahl)	1	0	0	1	1

Figura 7. Poziția bițiilor în sistemul de numerație Fracționare fără semn (Bit Position in dem Vorzeichenlose Bruchzahlen)

2.3 Vorzeichenlose Bruchzahlen (*Unsigned fractional*)

In DSP (Digital Verarbeitungssystem) und anderen Systemen ist es häufig erforderlich, Zahlen zu speichern, die sowohl eine Ganzzahl- als auch eine Bruch-komponente enthalten. Da einige Bitpositionen negativ sein können, kann das vorzeichenbehaftete fraktale System Zahlen speichern, die größer und kleiner als 1 sind. Der Stellenwert einer Ziffer im vorzeichenlosen Nummersystem ist $2^{Position}$, wobei die Position positiv oder negativ sein kann (siehe Figur 7). Daher ist das vorzeichenlose fraktale Nummerierungssystem eine Obermenge des vorzeichenlosen ganzzahligen Nummerierungssystems.

Tabelul 6 zeigt den Dezimalwert für jedes Bit in einer vorzeichenlosen gebrochenen Zahl.

Tabelul 7 zeigt, wie der Dezimalwert einer vorzeichenbehafteten gebrochenen Zahl bestimmt wird.

Tabelul 6. Ponderile bițiilor în cazul reprezentării în Fracționare fără semn a unui număr pe 5 biți (Vorzeichenlose gebrochene Werte)

Pozitia (Position)	Valoarea (Stellenwert)	Valoarea în zecimal (Dezimalwert)
-2	2^{-2}	0.25
-1	2^{-1}	0.50
0	2^0	1
1	2^1	2
2	2^2	4

Tabelul 7. Transformarea numerelor reprezentate în Fracționare fără semn în zecimal (Vorzeichenlose fraktionale Umwandlung)

Număr în Fracționare fără semn (Vorzeichenlose gebrochene Zahl)	Valoarea în zecimal (Dezimalwert)	Valoarea după conversia în zecimal (Umwandlung)
01001	2.25	$0 + 2^1 + 0 + 0 + 2^{-2} = 2 + 0.25 = 2.25$
11011	6.75	$2^2 + 2^1 + 0 + 2^{-1} + 2^{-2} = 4 + 2 + 0.5 + 0.25 = 6.75$
00010	0.5	$0 + 0 + 0 + 2^{-1} + 0 = 0.5$

Sistemul de numerație Fracționare fără semn folosește o notație convenabilă, pentru a urmări poziția virgulei (atât în binar, cât și în zecimal). Un număr care are N biți la stânga virgulei și M biți la dreapta acesteia este un număr $N.M$ (de exemplu, dacă are 8 biți la stânga virgulei și 3 biți la dreapta, se spune că este un număr 8.3).

Dacă toate numerele din sistemul de numerație au aceeași valoare a lui M (adică partea fracționară este reprezentată pe același număr de biți), atunci operațiile aritmetice sunt foarte simplu de realizat. De exemplu, se poate aduna un număr 12.3 cu un număr 8.3 folosind un circuit de adunare pe 15 biți. În **Figura 8** se prezintă modul în

Vorzeichenlose gebrochene Zahlen verwenden eine bequeme Notation, um die Position eines Radixpunkts (d. h. des Binär- oder Dezimalpunkts) zu verfolgen; Eine Zahl mit N Bits links vom Radixpunkt und M Bits rechts ist eine $N.M$ -Nummer (z. B. hat eine 8.3-Nummer 8 Ziffern links vom Radixpunkt und 3 Ziffern rechts).

Wenn alle Zahlen in Ihrem System den gleichen M -Wert haben (d. H. Dieselbe gebrochene Bitbreite), sind arithmetische Operationen unkompliziert. Beispielsweise können Sie einer 12.3-Nummer eine 8.3-Nummer hinzufügen, indem Sie einen 15-Bit-Binärraddierer verwenden. **Figura 8** zeigt, wie 00100101.101 (37.625) zu

care se realizează adunarea numerelor
 00100101.101 (37.625) și
 001101110011.001 (883.125).

001101110011.001 (883.125) hinzugefügt wird.

37.625	000000100101.101
+883.125	$+001101110011.001$
920.750	001110011000.110

Figura 8. Adunarea în cazul numerelor cu aceeași valoare a lui M (Addition mit den gleichen M-Werten)

Se observă că este necesar să se completeze cu 4 biți de 0, pentru ca virgula să ajungă pe aceeași poziție.

În cazul adunării unor numere cu valori diferite ale lui M , trebuie completat cu zerouri pentru a păstra virgulele aliniate. De exemplu, dacă se adună un număr 8.3 cu un număr 6.5, numărul 8.3 trebuie completat cu zerouri astfel încât să devină un număr 8.5. astfel, trebuie folosit un circuit sumator pe 13 biți în locul unuia pe 11 biți. În **Figura 8** se prezintă modul în care se realizează adunarea numerelor 11011011.110 (219.750) și 110111.11011 (55.84375).

Beachten Sie, dass Sie 4 Bits mit 0 eingeben müssen, damit das Komma an die gleiche Stelle kommt.

Um Zahlen mit unterschiedlichen M -Werten hinzuzufügen, müssen Sie zusätzliche Nullen hinzufügen, damit die Radixpunkte ausgerichtet bleiben. Um beispielsweise eine 8.3-Nummer zu einer 6.5-Nummer hinzuzufügen, müssen Sie die 8.3-Nummer mit Nullen auffüllen, um eine 8.5-Nummer zu erstellen. Daher müssen Sie einen 13-Bit-Nummernaddierer anstelle eines 11-Bit-Nummernaddierers verwenden, um die beiden Zahlen zu addieren. Abbildung 6 zeigt, wie 11011011.110 (219.750) zu 110111.11011 (55.84375) hinzugefügt wird.

219.75000	11011011.11000
+055.84375	+ 00110111.11011
275.59375	100010011.10011

Figura 9. Adunarea în cazul numerelor Fracționare fără semn cu valori diferențiale ale lui M. Biții introdusi în completare sunt evidențiați cu gri (Addieren mit unterschiedlichen M -Werten Vorzeichenlose Bruchzahlen. Die für das Auffüllen verwendeten Ziffern werden grau hervorgehoben)

În afară de alinierea virgulelor zecimale, nu există diferențe între sistemele de numerație. Întregi fără semn și Fracționare fără semn. De fapt, un număr Întreg fără semn este pur și simplu un număr Fracțional fără semn de forma $N.0$ (adică Întregii fără semn nu au deloc biți în partea fracțională). Orice

Abgesehen vom Ausrichten der Radixpunkte gibt es keinen Unterschied zwischen den vorzeichenlosen Ganzzahlsystemen und den vorzeichenlosen gebrochenen Nummerierungssystemen. Tatsächlich ist eine vorzeichenlose Ganzzahl einfach eine vorzeichenlose $N.0$ -Bruchzahl (d. H.

circuit hardware va funcționa atât pentru Întregi fără semn, cât și pentru Fracționare fără semn.

2.4 Fracționare în complementul lui 2 (*Two's Complement signed fractional*)

Acest sistem de numerație este similar cu sistemul de numerație Fracționare fără semn, în sensul că are o notație de tipul $N.M$ și virgulele trebuie aliniate în timpul operațiilor aritmetice.

Figura 10 prezintă cum se realizează adunarea unui număr 7.5 cu un număr 9.2 în acest sistem de numerație.

-36.28125		111011011.10111
+ 154.25000		+ 010011010.01000
<hr/>		<hr/>
+117.96875		001110101.11111

Figura 10. Adunarea numerelor Fracționare în complementul lui 2 cu valori diferite ale lui M. Biții introdusi în completare sunt evidențiați cu gri (Zweierkomplementierte fraktionierte Addition. Die für das Auffüllen verwendeten Ziffern werden grau hervorgehoben.)

2.5 Codul Gray

Codul Gray este un sistem de numerație care se folosește adeseori în domeniul aplicațiilor bazate pe senzori. Caracteristica principală a codului Gray este aceea că pe măsură ce înaintăm, un singur bit se modifică de-a lungul cuvintelor de cod.

Pentru a obține cuvintele codului Gray pe 2 biți începem de la cuvintele codului Gray pe 1 bit:

Pe doi biți vor fi patru cuvinte de cod binare. Primele două cuvinte de cod binare

Vorzeichenlose Ganzzahlen haben keine Bits rechts vom Radixpunkt). Jede Hardware, die für ganzzahlige Zahlen erstellt wurde, arbeitet mit gebrochenen Zahlen.

2.4 2-Komplement vorzeichen Bruchzahlen (*Two's Complement signed fractional*)

Wie das vorzeichenlose gebrochene Nummerierungssystem verwendet das mit zwei-Komplementen versehene gebrochene Nummerierungssystem eine $N.M$ -Notation, und die Radixpunkte müssen während arithmetischer Operationen ausgerichtet sein.

Abbildung 10 zeigt, wie eine 8.3-Nummer zu einer 5.5-Nummer hinzugefügt wird.

2.5 Gray Code

Gray Code ist ein Nummerierungssystem, das hauptsächlich in realen Sensoranwendungen verwendet wird. Das grundlegende Merkmal von Gray-Code ist, dass sich jeweils nur ein Bit ändert, während Sie die Zahlen nacheinander durchlaufen.

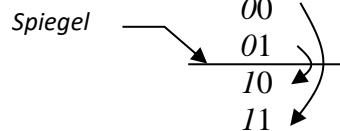
Für den Wörtern des 2-Bit-Gray zu erhalten, beginnen wir das Gray-Codewort mit 1 Bit:

0
1

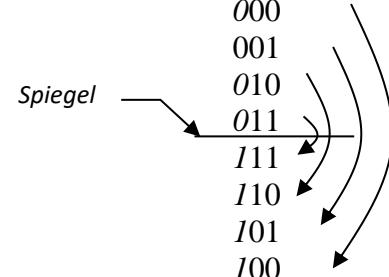
Zwei Bits sind vier binäre Codewörter. Die ersten beiden binären Codewörter des 2-Bit-

ale codului Gray pe 2 biți se obțin pur și simplu concatenând un '0' (pe poziția MSB) la cuvintele de cod Gray de 1 bit, după cum urmează (în *italice*):

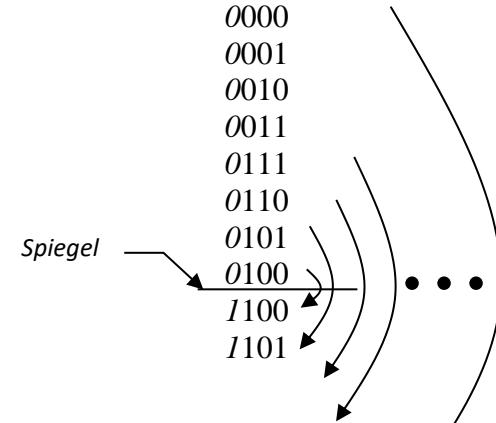
Ultimele două cuvinte de cod binare ale codului Gray pe 2 biți au un '1' pe poziția MSB. Apoi, ne imaginăm că există o oglindă care separată primele două cuvinte binare de ultimele. Cuvintele binare ale codului Gray pe 1 bit vor fi apoi reflectate în această oglindă, iar aceste reflexii se supun legilor fizicii: cuvintele care sunt mai aproape de oglindă apar ca fiind mai apropiate de suprafața ei, iar cuvintele care sunt mai îndepărtate vor părea că se reflectă mai adânc în oglindă:



Cuvintele codului Gray pe 3 biți se vor obține în aceeași manieră, pe baza Cuvintelor codului Gray pe 2 biți:



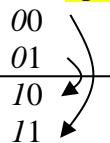
Cuvintele codului Gray pe 4 biți se vor obține în aceeași manieră, pe baza Cuvintelor codului Gray pe 3 biți:



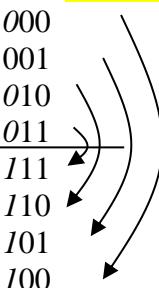
Gray-Codes werden einfach durch Verketten einer '0' (an der MSB-Position) mit den 1-Bit-Gray-Codewörtern wie folgt erhalten (in Kursivschrift):

00
01

Die letzten beiden Binärcodes des 2-Bit-Gray-Codes haben an der MSB-Position eine '1'. Dann stellen wir uns vor, dass es einen Spiegel gibt, der die ersten beiden binären Wörter von den letzten trennt. Die Binärwörter des 1-Bit-Gray-Codes werden dann in diesem Spiegel reflektiert, und diese Reflexionen unterliegen den Gesetzen der Physik: Die Wörter, die dem Spiegel am nächsten liegen, erscheinen näher an seiner Oberfläche, und die weiter entfernten Wörter scheinen zu sein spiegeln sich tiefer im Spiegel:



Die Wörter des 3-Bit-Gray-Codes werden auf die gleiche Weise erhalten, basierend auf den 2-Bit-Gray-Wortwörtern:



Die Wörter des 3-Bit-Gray-Codes werden auf die gleiche Weise erhalten, basierend auf den 2-Bit-Gray-Wortwörtern:

0000
0001
0010
0011
0111
0110
0101
0100
1100
1101

*I110
I111
I011
I010
I001
I000*

2.6 Mărime și semn (*Signed magnitude*)

Numerele reprezentate în Mărime și semn sunt utile în aplicații în care semnul și mărimea unui număr trebuie stocate separat. În acest sistem de numerație MSB reprezintă semnul numărului (0 = pozitiv, 1 = negativ) iar toți ceilalți biți reprezintă mărimea. Această notație este similară cu notația zecimală, unde se utilizează semnele + și - ca indicatori ai semnului și restul bițiilor se utilizează pentru reprezentarea mărimii.

Sistemele de numerație Mărime și semn și Complementul lui 2 folosesc ambele MSB pentru a determina semnul. Însă ele nu trebuie confundate. Deși în MSB poate fi utilizat pentru a determina semnul, ceilalți biți nu reprezintă mărimea, atunci când semnul este negativ. În plus, Complementul lui 2 nu are decât o reprezentare a lui zero, în timp ce sistemul de numerație Mărime și semn are două (+0 și -0).

Pentru a efectua operații aritmetice complexe pe date reprezentate în Mărime și semn, de regulă cel mai ușor este ca numerele să fie convertite în Complementul lui 2, ca operația să fie efectuată și apoi numerele să fie convertite înapoi în Mărime și semn.

2.6 Vorzeichenbehaftete Größenordnung (*Signed magnitude*)

Vorzeichenbehaftete Größen sind nützlich für Anwendungen, bei denen auf Vorzeichen und Größe einer Zahl separat zugegriffen werden muss. In Systemen mit vorzeichenbehafteten Beträgen repräsentiert das MSB das Vorzeichen der Zahl (d. H. 0 = positiv, 1 = negativ) und alle anderen Bits repräsentieren die Größe. Diese Schreibweise ähnelt der dezimalen Schreibweise, die ein + und - als Vorzeichenbit verwendet und die verbleibenden Bits zur Darstellung der Größe verwendet.

Das vorzeichenbehaftete und das Zweierkomplement-Nummerierungssystem verwenden beide das MSB, um das Vorzeichen zu bestimmen. Verwechseln Sie die vorzeichenbehaftete Größe jedoch nicht mit Zweierkomplement. Obwohl das MSB zur Bestimmung des Vorzeichens in Zweierkomplement verwendet werden kann, repräsentieren die anderen Bits nicht die Größe, wenn das Vorzeichen negativ ist. Zusätzlich hat das Zweierkomplement nur eine Darstellung von Null, während das vorzeichenbehafteten Nummerierungssystem hat zwei (d.h. +0 und -0).

Um komplexe arithmetische Operationen mit vorzeichenbehafteten Betragsdaten durchzuführen, ist es in der Regel einfacher, die Zahlen in Zweierkomplement umzuwandeln, die Operationen auszuführen und die Zahlen dann gegebenenfalls wieder in ein vorzeichenbehaftetes Größenordnungssystem umzuwandeln.

2.7 Complementul lui 2 cu deplasament (*Offset two's Complement*)

Sistemul de numerație Complementul lui 2 cu deplasament este foarte folosit în convertoarele analog/digitale și digital/analoge. Caracteristica distinctivă constă în aceea că numerele de la -4 la 3 se succed monoton, aşa cum numărăm în binar de la 000 la 111. Spre deosebire de aceasta, valorile zecimale ale numerelor reprezentate în Complementul lui 2 merg de la 0 la 3 și apoi de la -4 la -1 pe măsură ce numărul binar crește de la 000 la 111.

Pentru a converti numerele din Complementul lui 2 în Complementul lui 2 cu deplasament este suficient să inversăm MSB, după cum se vede în **Tabelul 8**.

Tabelul 8. Conversia numerelor în Complementul lui 2 cu deplasament (Umwandeln des Offset-Zweierkomplement)

Zecimal (Dezimalwert)	Complementul lui 2 (Zweierkomplement)	Complementul lui 2 cu deplasament (Offset-Zweierkomplement)
-4	100	000
-3	101	001
-2	110	010
-1	111	011
0	000	100
1	001	101
2	010	110
3	011	111

2.8 Complementul lui 1 (*One's Complement*)

Sistemul de numerație Complementul lui 1

2.7 Offset Zweierkomplement (*Offset two's Complement*)

Das Zwei-Offset-Komplement-Nummerierungssystem wird von vielen D / A- und A / D-Wandlern verwendet. Das Unterscheidungsmerkmal dieses Nummerierungssystems besteht darin, dass sich die Zahlen von -4 nach 3 monoton bewegen, wenn Sie binär von 000 bis 111 zählen - es gibt keine Sprünge oder Unterbrechungen. Im Gegensatz dazu geht der Dezimalwert der Zweierkomplementzahlen von 0 bis 3 und zählt dann von -4 bis -1, wenn die Binärzahl von 000 bis 111 fortschreitet.

Um Zahlen von Zweierkomplement in Offset-Zweierkomplement umzuwandeln, invertieren Sie einfach das MSB, wie in **Tabelul 8** gezeigt.

2.8 Einerkomplement (*One's Complement*)

Das

Einerkomplement-

este folosit mai rar, deoarece prezintă aceleasi dezavantaje ca toate sistemele de numerație nebazate pe Complementul lui 2: este greu să se efectueze operații aritmetice cu el și are două reprezentări ale lui 0. Când adunăm un număr cu opusul lui în Complementul lui 1, rezultatul nu este 0, ceea ce generează o algebră inconsistentă. Există două reprezentări ale lui 0, 000 și 111, ceea ce face mai dificilă detecția lui zero (trebuie realizată o operație SAU pe N biți și o operație SI pe N biți). Pentru a nega un număr reprezentat în Complementul lui 1 trebuie să inversăm toți biții (nu se mai adună 1 ca în cazul Complementului lui 2).

Nummerierungssystem wird selten verwendet, da es die gleichen Nachteile wie alle Nicht-Zweierkomplement-Nummern hat - es ist schwierig, arithmetische Operationen auszuführen, und es hat zwei Repräsentationen von Null (d. H. +0 und -0). Wenn Sie zu ihrer Umkehrung eine Ein-Komplement-Nummer hinzufügen, ist die Antwort nicht Null, wodurch eine inkonsistente Algebra entsteht. Es gibt beiden Darstellungen von Null 000 und 111, was die Nullpunkt erfassung schwieriger macht (d. H. Sie müssen ein N -Bit-ODER und ein N -Bit-UND-Gatter verwenden). Um eine Zahl in dem EinerKomplement-Nummerierungssystem zu negieren, invertieren Sie einfach alle Bits (d. H. Sie fügen keine 1-Zahl wie in Zweierkomplement hinzu).

2.9 Virgulă mobilă (*Floating point*)

Reprezentarea numerelor în virgulă mobilă este adeseori necesară în domeniul științific și tehnic mai ales pentru numere mari. În limbajele de programare este pus la dispoziția programatorului tipul de date REAL. Însă această denumire este înselătoare, deoarece aritmetică în virgulă mobilă permite cel mult reprezentarea aproximativă a numerelor reale. Pentru o mai bună evaluare a limitelor și posibilităților reprezentărilor bazate pe virgula mobilă prezentăm o serie de aspecte legate de reprezentarea numerelor conform standardelor IEEE.

Reprezentarea numerelor în virgulă mobilă Z se bazează pe așa-numita **reprezentare semi-logaritmică**, des întâlnită în domeniul tehnic, în care *mantisa* și *exponentul* sunt codificate ca vectori de biți:

2.9 Gleitkomma-Zahlen (*Floating point*)

Gleitkomma-Zahlen werden im wissenschaftlich-technischen Bereich insbesondere bei großen Zahlenbereichen vielfach benötigt. In den Programmiersprachen wird dem Programmierer der REAL-Datentyp zur Verfügung gestellt. Diese Bezeichnung ist aber irreführend, denn die verfügbare Gleitkomma-Arithmetik erlaubt höchstens die näherungsweise Darstellung von reellen Zahlen. Zwecks besserer Einschätzung der Grenzen und Möglichkeiten üblicher Gleitkomma-Darstellungen wollen wir hier v.a. auf die Darstellung nach IEEE-Standards eingehen.

Die Darstellung von Gleitkomma-Zahlen Z basiert auf der im technischen Bereich üblichen **halblogarithmischen Darstellung**, deren Mantisse und Exponent als Bitvektoren kodiert werden:

$$Z = (-1)^{VZ_M} \times M \times 2^{E'}$$

M: valoarea mantisei (Betrag der Mantisse)

VZ_M: semnul mantisei (Vorzeichen der Mantisse)

E' = (VZ_E, E): Exponent

Formatele cele mai comune sunt următoarele:

Übliche Formate sind die folgenden:

a)	VZ_M	VZ_E	E	<i>MANTISSENBETRAG M</i>
----	--------	--------	-----	--------------------------

Unde: $VZ_M = Bitul\ de\ semn\ al\ mantisei\ M;$
 $VZ_E = Bitul\ de\ semn\ al\ exponentului\ E.$

Mit: $VZ_M = Vorzeichen\ zu\ M;$ $VZ_E = Vorzeichen\ zu\ E.$

b)	VZ_M	<i>CHARAKTERISTIK C</i>	<i>MANTISSENBETRAG M</i>
----	--------	-------------------------	--------------------------

Caracteristica se calculează adăugând magnitudinea celui mai mic exponent dorit la toți exponentii.

Acest lucru permite practic lucrul cu exponenti exclusiv pozitivi. Astfel se realizează simplificarea multor algoritmi. Numai în momentul conversiei în reprezentarea externă se scade din nou numărul.

Standardul IEEE de reprezentare a numerelor reale propune trei moduri de reprezentare a numerelor reale:

- Formatul scurt pe 4 octeți (*simplă precizie*)
- Formatul lung pe 8 octeți (*dublă precizie*)
- Formatul temporar pe 10 octeți (pentru coprocesorul matematic).

Bitul de semn VZ_M se reprezintă pe un singur bit. Caracteristica se reprezintă de regulă pe 8, 11, sau 15 biți la formatul scurt, lung, respectiv temporar. Mantisa se reprezintă de regulă pe 23, 52, respectiv 64 de biți.

Pentru fiecare reprezentare: VZ_M este 0 dacă numărul este pozitiv și 1 dacă numărul este negativ. Caracteristica $C = E + 7F_{16}$ (respectiv $3FF_{16}$ la dublă precizie și $3FFF_{16}$ la formatul temporar).

Pentru găsirea mantisei mai întâi se normalizează numărul scris în baza 2, adică se scrie numărul sub forma: $NR = 1.< \text{alte}$

Die Charakteristik wird berechnet, indem der Betrag des kleinsten gewünschten Exponenten auf alle Exponenten addiert. Dies erlaubt praktisch ein Arbeiten mit ausschließlich positiven Exponenten. Es bewirkt eine Vereinfachung vieler Algorithmen. Erst bei Wandlung in die externe Darstellung wird die addierte Zahl wieder abgezogen.

Der IEEE-Standard für die Darstellung reeller Zahlen schlägt drei Möglichkeiten zur Darstellung reeller Zahlen vor:

- Kurzes (4-Byte-Format) – *einfache Präzision*
- Langes (8-Byte-Format) – *doppelte Präzision*
- Temporäres 10-Byte-Format (für den mathematischen Coprozessor).

Das Vorzeichenbit VZ_M wird in einem einzelnen Bit dargestellt. Die Charakteristik ist normalerweise 8, 11 oder 15 Bit in Kurz-, Lang- oder temporärem Format. Mantissa ist in der Regel 23, 52 und 64 Bit normalerweise.

Für jede Darstellung: VZ_M ist 0, wenn die Zahl positiv ist, und 1, wenn die Zahl negativ ist. Die Charakteristik $C = E + 7F_{16}$ ($3FF_{16}$ für doppelte Präzision und $3FFF_{16}$ für temporäres Format).

Um die Mantisse zu finden, zuerst wird die in die Basis 2 geschriebene Zahl normalisiert, d.h. die Zahl wird in der

cifre binare $> \times 2^E$.

La reprezentarea în format IEEE *simplă precizie* și *dublă precizie*, mantisa este formată din cifrele de după virgulă, deci primul 1 dinaintea virgulei nu se mai reprezintă în mantisă, iar la formatul temporar se reprezintă toate cifrele din număr.

Exemple

Exemplul 1

Să se reprezinte în format IEEE *simplă precizie* numărul $(17.6)_{10}$.

Se va converti separat partea întreagă și cea zecimală și se obține:

Partea întreagă:

$$(17)_{10} = (11)_{16} = (0001\ 0001)_2$$

Partea zecimală: $(0.6)_{10} = (0.(1001))_2$ (se observă că numărul este periodic).

$$\text{Deci } (17.6)_{10} = 10001.(1001)_2.$$

Se normalizează numărul: $(17.6)_{10} = 10001 \cdot (0.6)_{10}$.
 $(1001)_2 = 1.0001(1001) \times 2^4$ (deși era mai corect în loc de 2^4 să se scrie $(10^{100})_2$ pentru că notarea era în baza 2; faptul că se calculează caracteristica mai ușor în hexazecimal decât în binar poate fi o scuză).

Din această reprezentare se poate deduce mantisa (ceea ce este după virgulă, deci fără acel 1 dinaintea virgulei care prin convenție nu se mai reprezintă) și anume:

$$M = 0001(1001)_2.$$

În continuare se calculează caracteristica:

$$C = E + 7F_{16} = 4 + 7F_{16} = 83_{16} = (1000\ 0011)_2$$

Se va scrie bitul semn 0 și deja se poate trece la scrierea reprezentării:

folgenden Form geschrieben: NR = 1.
<andere Binärziffern> $\times 2^E$.

Bei der Darstellung im IEEE *einfache Präzision* und *doppelte Präzision* besteht die Mantisse nachher die Kommas nummerierten Ziffern, sodass die erste 1 vor dem Komma nicht mehr in der Mantisse enthalten sind. Im temporären Format alle Zahlen in der Zahl enthalten sind.

Beispiele

Beispiel 1

Stellen Sie in einem kurzen IEEE *einfache Präzision* Format die Nummer $(17.6)_{10}$ dar.
Es wird das Ganze und den Dezimalteil getrennt konvertieren und erhalten:

Ganzer Teil:

$$(17)_{10} = (11)_{16} = (0001\ 0001)_2$$

Dezimalteil: $(0.6)_{10} = (0.(1001))_2$ (es wird angemerkt, dass die Zahl periodisch ist).

$$\text{Also } (17.6)_{10} = 10001.(1001)_2.$$

Normalisieren Sie die Zahl: $(17.6)_{10} = 10001 \cdot (0.6)_{10} = 1.0001(1001) \times 2^4$ (obwohl es mehr war richtig statt 2^4 schreiben $(10^{100})_2$, weil die Note im Basis 2 war; Die Tatsache, dass die Charakteristik einfacher hexadezimal als binär zu berechnen ist, kann eine Entschuldigung sein).

Aus dieser Darstellung können wir die Mantisse ableiten (die nach dem Komma steht, d.h. ohne die 1 vor dem Komma, die laut Konvention nicht mehr steht), nämlich:

$$M = 0001(1001)_2.$$

Als nächstes wird die Charakteristik berechnet:

Es wird das Vorzeichenbit 0 geschrieben und Sie können bereits zum Schreiben der Darstellung wechseln:

Semn (Vorzeichenbit)	Charakteristik	Mantissa
0	10000011	0001100110011001100

Rezultatul final al reprezentării este: $(41\ 8C\ CC\ CC)_{16}$.

În cazul practic, în memoria calculatorului, datorită unei rotunjiri care se face la ultimul bit al reprezentării, se poate observa că în mantisă ar mai urma un 1 după cei 23 de biți, iar calculatorul va face rotunjire superioară, de aceea pe ultimul bit (LSB) va apărea 1, iar reprezentarea va fi: $(41\ 8C\ CC\ CD)_{16}$.

Exemplul 2

În mod analog se va reprezenta $(-23.5)_{10}$:

$$(23)_{10} = (17)_{16} = (1\ 0111)_2$$

$$(0.5)_{10} = (0.1)_2$$

Deci $(23.5)_{10} = 10111.1 = 1.01111 \times 2^4$, de unde rezultă $M = 0111100000000000\dots$ (23 de biți).

Caracteristica $C = 7F_{16} + 4_{16} = 83_{16}$.

Se pune bitul semn pe 1.

Reprezentarea direct în hexazecimal este $C1\ BC\ 00\ 00_{16}$.

În continuare se pune problema inversă reprezentării: se dă reprezentarea unui număr în format IEEE simplă precizie și se cere aflarea numărului real care este astfel reprezentat.

Exemplul 3

Se dă reprezentarea 43 04 33 3316 și se cere să se obțină valoarea zecimală a numărului real reprezentat.

Se scrie în binar reprezentarea: $(0100\ 0011\ 0000\ 0100\ 0011\ 0011\ 0011\ 0011)_2$. De aici se deduce că:

- Semnul este 0.

- Caracteristica $C = 1000\ 0110_2 = 86_{16}$.

Rezultă deci că exponentul $E = 86_{16} - 7F_{16} = 7_{16}$.

- Mantisa $M = 0000\ 1000\ 0110\ 0110\dots$

Numărul este:

$$Nr = 1.M \times 2^E = (1.0000\ 1000\ 0110\dots)_2 \times 2^7$$

Das Endergebnis der Darstellung ist: $(41\ 8C\ CC\ CC)_{16}$.

In der Praxis im das Speicher von dem Computer, aufgrund der Rundung beim letzten Bit von der Darstellung, kann man sehen, dass die Mantisse ein 1 nach dem 23-Bit folgen würden und der Computer wird oben abgerundet sein, so das letzte Bit (LSB) wird 1 erscheinen, und die Darstellung wird: $(41\ 8C\ CC\ CD)_{16}$.

Beispiel 2

Analog wird die Darstellung von $(-23.5)_{10}$ sein:

$$(23)_{10} = (17)_{16} = (1\ 0111)_2$$

$$(0.5)_{10} = (0.1)_2$$

Also $(23.5)_{10} = 10111.1 = 1.01111 \times 2^4$, was zu $M = 0111100000000000\dots$ (23 Bits) führt.

Die Charakteristik $C = 7F_{16} + 4_{16} = 83_{16}$.

Setzen Sie das Vorzeichenbit auf 1.

Die direkte hexadezimale Darstellung ist $C1\ BC\ 00\ 00_{16}$.

Weitere umkehren die Frage der Darstellung: Darstellung gegeben wird eine Zahl in IEEE einfacher Präzision und die reelle Zahl erfordert zu finden ist so dargestellt.

Beispiel 3

Die Darstellung 43 04 33 3316 ist gegeben und es ist erforderlich, den Dezimalwert der dargestellten reellen Zahl zu erhalten.

Die Darstellung ist binär geschrieben: $(0100\ 0011\ 0000\ 0100\ 0011\ 0011\ 0011\ 0011)_2$.

Daraus folgt:

- Das Vorzeichen ist 0.

- Die Charakteristik $C = 1000\ 0110_2 = 86_{16}$.

Daraus folgt der Exponent $E = 86_{16} - 7F_{16} = 7_{16}$.

- Die Mantissa $M = 0000\ 1000\ 0110\ 0110\dots$

Die Zahl ist:

$$Nr = 1.M \times 2^E = (1.0000\ 1000\ 0110\dots)_2 \times 2^7$$

$$\begin{aligned}
 &= (1000\ 0100.00110011\dots)_2 \\
 &= 128 + 4 + 0.125 + 0.0625 + \dots \\
 &\approx 132.1875
 \end{aligned}$$

Valoarea exactă era 132.2.

Utilizarea Caracteristicii prezintă, de asemenea, avantajul de a permite efectuarea comparațiilor de magnitudine pentru numerele în reprezentate în virgulă mobilă folosind exact comenzile corespunzătoare pentru compararea numerelor naturale sau reprezentate în Complementul lui 2.

$$\begin{aligned}
 &= (1000\ 0100.00110011\dots)_2 \\
 &= 128 + 4 + 0,125 + 0,0625 + \dots \\
 &\approx 132,1875
 \end{aligned}$$

Der genaue Wert war 132,2.

Die Benutzung der Charakteristik hat weiter den angenehmen Effekt, dass betragmäßige Größenvergleichen für Gleitkommazahlen mit den entsprechenden Befehlen für natürliche oder 2er-Komplement-Zahlen ausgeführt werden können.

3. Reprezentări binare și ordini de plasare

3.1 Dimensiunea reprezentării

În vorba de calcule efectuate cu o mașină, există o serie de restricții legate de reprezentarea numerelor. Cea mai importantă dintre ele este *dimensiunea reprezentării*, adică numărul maxim de biți din reprezentarea unui număr. Să notăm cu n această dimensiune de reprezentare. Numărul n este o constantă a sistemului de calcul, stabilită la proiectarea acestuia.

Valorile cele mai uzuale ale lui n la sistemele de calcul actuale sunt: 8, 16, 32 și 64.

Tot în categoria restricțiilor intră și faptul că dimensiunile celor doi operanzi care participă la o operație, precum și dimensiunile rezultatului sunt de asemenea constante ale calculatorului, indiferent de tipul de codificare a numerelor. Pentru a preciza regulile de dimensionare în operațiile binare asupra numerelor întregi, vom folosi sintagma "operație pe n biți". Regulile de dimensionare în urma operațiilor sunt următoarele:

a) Operațiile de adunare pe n biți și scădere pe n biți presupun că ambii termeni sunt reprezentați pe câte n biți, iar rezultatul (suma sau diferența) se va reprezenta tot pe n biți, vezi Figura 11.

3. Binäre Darstellungen und Platzierungsaufträge

3.1 Darstellungsdimension

In Bezug auf Maschinenberechnungen gibt es eine Reihe von Einschränkungen bei der Darstellung von Zahlen. Das wichtigste davon ist die *Größe der Darstellung*, dh die maximale Anzahl von Bits in der Darstellung einer Zahl. Notieren wir mit n diesen diese Dimension der Repräsentation. Die Zahl n ist eine Konstante des Berechnungssystems, die bei seiner Auslegung festgelegt wurde.

Die häufigsten Werte von n in aktuellen Computersystemen sind: 8, 16, 32 und 64.

Ebenfalls in die Kategorie der Einschränkungen fällt die Tatsache, dass die Dimensionen der beiden an einer Operation beteiligten Operanden sowie die Größe des Ergebnisses unabhängig von der Art der Codierung der Zahlen auch für den Computer konstant sind. Um Größenregeln in Binäroperationen für Ganzzahlen festzulegen, verwenden wir den Ausdruck „ n -Bit-Operation“. Die Größenregeln für folgende Vorgänge lauten wie folgt:

a) N -Bits-Addition und N -Bits-Subtraktionsoperationen setzen voraus, dass beide Terme durch n Bits dargestellt werden und das Ergebnis (Summe oder Differenz) auch durch n Bits dargestellt wird, siehe Figura 11.

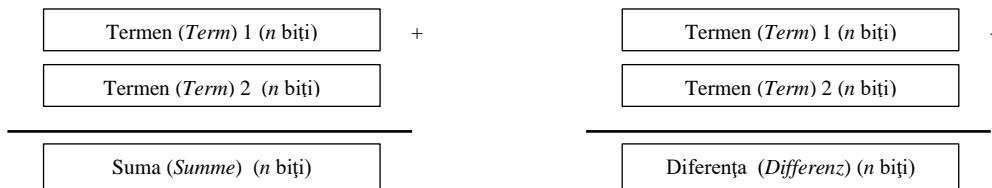


Figura 11. Dimensiuni de reprezentare la adunare și scădere (Dimensionen der Darstellung für die Addition und die Subtraktion)

b) Înmulțirea pe n biți presupune că ambii factori sunt reprezentați pe câte n biți, iar produsul lor va fi reprezentat pe $2 \times n$ biți, vezi **Figura 12**.

b) n -Bits Multiplikation setzt voraus, dass beide Faktoren durch n Bits dargestellt werden und ihr Produkt durch $2 \times n$ Bits dargestellt wird, siehe **Figura 12**.

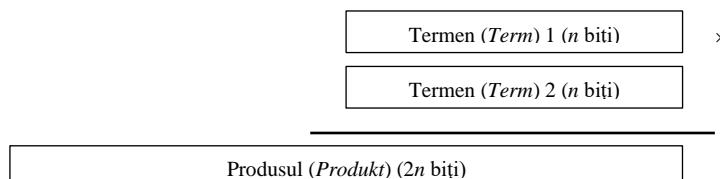


Figura 12. Dimensiuni de reprezentare la înmulțire (Dimensionen der Darstellung für die Multiplikation)

c) împărțirea pe n biți (oarecum invers față de înmulțire), impune condiția ca deîmpărțitul să fie reprezentat pe $2 \times n$ biți, iar împărțitorul pe n biți. Operația furnizează două rezultate: câtul reprezentat pe n biți și restul reprezentat tot pe n biți, vezi **Figura 13**.

c) n -Bits Division (etwas entgegengesetzt zur Multiplikation) impliziert, dass der Dividend auf $2 \times n$ Bits und der Divisor auf n Bits dargestellt wird. Die Operation liefert zwei Ergebnisse: den Quotient, der auf n Bits dargestellt ist, und den Rest, der ebenfalls auf n Bits dargestellt ist, siehe Figura 13.

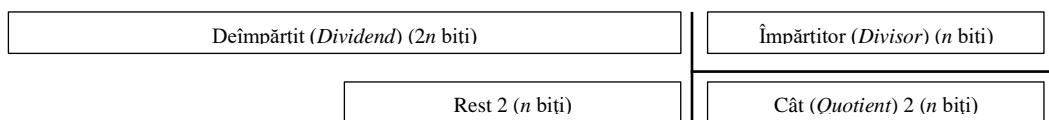


Figura 13. Dimensiuni de reprezentare la împărțire (Dimensionen der Darstellung für die Division)

Dacă rezultatul unei operații nu încape în dimensiunea de reprezentare, atunci se vor pierde biții cei mai semnificativi, rămânând biții mai puțin semnificativi: 0, 1, 2 și.a.m.d., calculatorul semnalând fenomenul de *depășire*.

Wenn das Ergebnis einer Operation nicht in die Repräsentationsdimension passt, gehen die höchswertigen Bits verloren, sodass die weniger bedeutend Bits bleiben: 0, 1, 2 usw. Der Computer signalisiert das *Überlaufphänomen*.

3.2 Organizarea și memorarea datelor

Atât pentru reprezentarea întregilor, cât și pentru reprezentarea altor tipuri de date,

3.2 Daten organisieren und speichern

Jedes Computersystem verwendet sowohl für die Ganzzahlen Darstellung als auch für

orice sistem de calcul folosește o componentă specială, numită unitate de memorie. Prezentăm principalele elemente de structurare a acesteia.

Unitatea elementară de informație este bitul. Într-un bit se poate reprezenta o informație care poate să aibă doar două valori posibile: 0 sau 1. Interpretarea acestor valori se realizează în funcție de context: un bit poate să însemne 0 sau 1, *true* sau *false*, bărbat sau femeie, bine sau rău, alb sau negru etc. Din punct de vedere tehnologic, un bit este materializat prin niveluri de tensiune în curent continuu: 0 volți poate însemna „0”, în timp ce o tensiune de +5V poate însemna „1”.

Definiție: Un octet (*byte*) este o succesiune de 8 biți, numerotată de la 0 la 7, ca în **Figura 14**.

7	6	4	5	3	2	1	0
Bit high (MSB)				Bit low (LSB)			

Figura 14. Numerotarea bițiilor în cadrul unui octet (Nummerierung der Bits in einem Byte)

Octetul este unitatea elementară de adresare a memoriei. Fiecare octet are atașat un număr întreg și nenegativ numit *adresa octetului respectiv*. Primul octet din memorie are adresa 0, al doilea octet are adresa 1, al treilea are adresa 2 și.a.m.d. Sistemul poate referi / identifica fiecare octet din memorie folosind adresa acestuia. Intuitiv, ne putem imagina memoria ca o mulțime de căsuțe (așa cum sunt cele de la post-restant). Căsuțele sunt numerotate începând de la 0 și în fiecare căsuță poate fi un singur număr. În momentul în care depunem în căsuță un alt număr, vechiul număr se pierde (ca și la înregistrările audio / video pe bandă, rămâne doar ultima înregistrare). Numărul de pe ușa căsuței reprezintă adresa / referința, iar numărul din căsuță reprezintă *conținutul*.

die Darstellung anderer Datentypen eine spezielle Komponente, die als Speichereinheit bezeichnet wird. Wir präsentieren die Hauptelemente der Strukturierung.

Die elementare Informationseinheit ist das Bit. Ein Bit kann Informationen darstellen, die nur zwei mögliche Werte haben können: 0 oder 1. Die Interpretation dieser Werte erfolgt über den Kontext: Ein Bit kann 0 oder 1 bedeuten, richtig oder falsch, männlich oder weiblich, gut oder schlecht, weiß oder schwarz etc. Aus technologischer Sicht wird ein Bit durch Gleichspannungspegel materialisiert: 0 Volt kannst '0' bedeuten, während eine Spannung von +5 V '0' bedeuten kann.

Definition: Ein Byte (Byte) ist eine 8-Bit-Sequenz, die wie in Figura 14 von 0 bis 7 nummeriert ist.

Das Byte ist die grundlegende Speicheradressiereinheit. Jedes Byte hat eine ganz und eine nicht negative Zahl, die als Adresse dieses Bytes bezeichnet wird. Das erste Byte des Speichers hat die Adresse 0, das zweite Byte hat die Adresse 1, das dritte Byte hat die Adresse 2 und so weiter. Das System kann jedes Byte des Speichers anhand seiner Adresse referenzieren / identifizieren. Intuitiv können wir uns das Speicher als viele Hütten vorstellen (wie sie aus der Nachruhe stammen). Die Kästchen sind beginnend mit 0 nummeriert und jedes Kästchen kann eine einzelne Nummer sein. Wenn wir eine andere Nummer in die Box legen, geht die alte Nummer verloren (wie bei Audio- / Videobändern bleibt es nur die letzte Aufzeichnung). Die Nummer an der Tür der Box ist die Adresse / Referenz, und

Referirea la un octet se face, deci, prin adresa lui. De multe ori se practică referirea la un octet nu prin adresa lui, ci prin poziția lui față de un alt octet. În primul caz vorbim de *adresa absolută* a unui octet, iar în al doilea caz vorbim de *adresa relativă* față de un alt octet. În al doilea caz adresa absolută se obține adunând la adresa octetului de referință adresa relativă. De exemplu, dacă un octet A are adresa absolută 5643 și un octet B are adresa relativă 5 față de A, atunci adresa absolută a octetului B este $5643 + 5 = 5648$. Ca terminologie, spunem că octetul B este cu 5 octeți mai la dreapta decât A. Vezi în **Figura 15** ilustrarea acestei situații.

die Nummer in der Box ist der *Inhalt*.

Der Verweis auf ein Byte erfolgt durch seinem Adress. Es wird oft verwendet, um auf ein Byte nicht durch seine Adresse, sondern durch seine Position zu einem anderen Byte zu verweisen. Im ersten Fall beziehen wir uns auf die *absolute Adresse* eines Bytes und im zweiten Fall in Bezug auf die *relative Adresse* eines anderen Bytes. Im zweiten Fall wird die absolute Adresse erhalten, indem die relative Adresse zur Referenzbyteadresse addiert wird. Wenn zum Beispiel ein Byte A eine absolute Adresse von 5643 hat und ein Byte B eine relative Adresse von 5 in Bezug auf A hat, dann ist die absolute Adresse von Byte B $5643 + 5 = 5648$. Als Terminologie sagen wir, dass Byte B 5 Bytes rechts ist als A. Siehe Figura 15 zur Veranschaulichung dieser Situation.

Conținut (<i>Inhalt</i>) octet 0 Adresa 0	Conținut (<i>Inhalt</i>) octet 1 Adresa 1	Conținut (<i>Inhalt</i>) octet 2 Adresa 2	...	Conținut (<i>Inhalt</i>) octet 5643 Adresa 5643	...	Conținut (<i>Inhalt</i>) octet 5648 Adresa 5648	...
---	---	---	-----	---	-----	---	-----

Figura 15. Succesiunea octetilor în memorie (Reihenfolge der Bytes im Speicher)

Bitul 0 al octetului se numește bitul cel mai puțin semnificativ (LSB), bitul de rang minim, bitul cel mai din dreapta, bitul *low* etc. Bitul 7 este bitul cel mai semnificativ (MSB), bitul de rang maxim, bitul cel mai din stânga, bitul *high* etc.

Să clarificăm două noțiuni fundamentale: **adresă - adresare** și **conținut - numerotare conținut**. În ceea ce privește adresarea: este unanim acceptată convenția că adresele octetilor în memorie cresc de la stânga la dreapta (**Figura 15**); deci la adresare relativă, octetul de referință este la stânga octetului curent. În ceea ce privește conținutul unei locații de memorie: numerotările entităților dintr-un conținut se fac de la dreapta spre stânga, aşa cum am văzut la numerotarea biților unui octet (**Figura 14**).

Das Bit 0 des Bytes wird das niedrigstwertige Bit (LSB), das niedrigste Bit, das Bit ganz rechts, das *low* Bit usw. genannt. Bit 7 ist das höchstwertige Bit (MSB), das Bit mit dem höchsten Rang, das Bit ganz links, das *high* Bit usw.

Lassen Sie uns zwei grundlegende Begriffe klarstellen: **Adresse - Adressierung** und **Inhalt - Nummerierung von Inhalt**. In Bezug auf die Adressierung gilt allgemein die Konvention, dass die Adressen der Bytes im Speicher von links nach rechts zunehmen (Figura 15). Im Falle von relativen Adressierung befindet sich das Referenzbyte links vom aktuellen Byte. Zum Inhalt eines Speicherorts: Die Nummerierung der Entitäten in einem Inhalt erfolgt von rechts nach links, wie wir an der Nummerierung der Bits eines Bytes gesehen haben (Figura

Entitatea octet este folosită practic de către toate instrucțiunile de prelucrare și de schimb cu exteriorul ale unui sistem de calcul. În contextul comunicațiilor trebuie reținut postulatul că: octetul are aceeași reprezentare, indiferent de sistemul de calcul. Cu alte cuvinte, se spune că octetul este portabil.

Accesul la biții unui octet se poate face prin intermediul unor instrucțiuni specializate. În particular, sunt situații în care se folosește ca entitate de execuție semioctetul (*nibble*). Un semioctet este format din patru biți, alăturați. Vorbim de semioctet low, sau semioctet mai puțin semnificativ, sau semioctet drept, sau cifră hexazecimală dreaptă, sau *nibble* drept etc. Analog, vorbim de semioctet high, sau semioctet semnificativ, sau semioctet stâng, sau cifră hexazecimală stângă, sau *nibble* stâng etc. Schematic, această împărțire apare ca în **Figura 16.**



Figura 16. Semioceții componenți ai unui octet (Halb-Byte-Komponenten eines Bytes)

Prelucrările fundamentale sunt efectuate pe octeți și pe grupuri de octeți consecutivi. În particular, operațiile cu numere întregi se pot efectua fie pe octeți, fie pe grupuri de octeți consecutivi.

O succesiune de octeți consecutivi de dimensiune fixată, privită ca o entitate de sine stătătoare formează o **locație** sau **unitate de prelucrare**.

Adresa unei locații este egală cu adresa primului octet component al locației (cu cea mai mică adresă a octetilor ce o compun).

Dimensiunea unei locații este egală cu numărul de octeți care o compun.

14).

Die Byte-Entität wird praktisch von allen Verarbeitungs- und Austauschanleitungen mit der Außenseite eines Computersystems verwendet. Im Zusammenhang mit der Kommunikation ist zu beachten, dass das Byte unabhängig vom Rechensystem die gleiche Darstellung hat. Mit anderen Worten, das Byte soll portabel sein.

Der Zugriff auf die Bits eines Bytes kann durch spezielle Anweisungen erfolgen. Insbesondere gibt es Situationen, in denen das Halbbyte (*Nibble*) als Ausführungsentität verwendet wird. Ein Halbbyte besteht aus vier Bits, die zusammengefügt werden. Wir sprechen von Low Halbbyte, oder niedrigstwertige Halbbyte, oder rechter Halbbyte, oder rechter Hexadezimale Ziffer, oder rechter *Nibble* etc. In analoger Weise sprechen wir von High Halbbyte, oder höchstwertige Halbbyte, oder linker Halbbyte, oder linker hexadezimaler Ziffer, oder linker *Nibble* usw. Schematisch erscheint diese Unterteilung wie in Figura 16.

Grundlegende Verarbeitung werden von Bytes und aufeinanderfolgenden Bytegruppen ausgeführt. Insbesondere können Ganzzahloperationen entweder byteweise oder am aufeinanderfolgende Bytegruppen ausgeführt werden.

Eine Reihe aufeinanderfolgender Festgröße-Bytes, die als eigenständige Entität betrachtet werden, bilden einen **Speicherort** oder eine **Verarbeitungseinheit**.

Die Adresse eines Speicherorts entspricht der Adresse des ersten Bytes des Speicherorts (mit der kleinsten Byteadresse von sein Byten).

Die Größe eines Speicherorts entspricht der Anzahl der von ihm Bytes.

Dimensiunea și denumirea pe care o poartă o locație, variază de la un tip de sistem de calcul la altul. De exemplu, la unele sisteme, cum ar fi cele din familia IBM-PC:

- doi octeți consecutivi formează un *cuvânt*;
- patru octeți consecutivi formează un *dublucuvânt*.

La astfel de sisteme vorbim de locații octet, locații cuvânt și locații dublucuvânt. În **Figura 17** prezentăm un cuvânt cu subdiviziunile (sub-entitățile) lui, iar în **Figura 18** prezentăm un dublucuvânt cu subdiviziunile (sub-entitățile) lui.

Die Größe und der Name eines Speicherorts variieren von einem Typ von Computersystem zu einem anderen. Beispiel: Auf einigen Systemen, z. B. in der IBM-PC-Familie:

- zwei aufeinanderfolgende Bytes bilden einen *Wort*;
- Vier aufeinanderfolgende Bytes bilden einen *Doppelwort*.

In solchen Systemen sprechen wir von Byte-Speicherorten, Wortspeicherorten und Doppelwortspeicherorten. In Figura 17 präsentieren wir einen Wort mit seinen Unterteilungen (Unter-Entitäten), und in Figura 18 präsentieren wir ein Doppelwort mit seinen Unterteilungen (Unter-Entitäten).

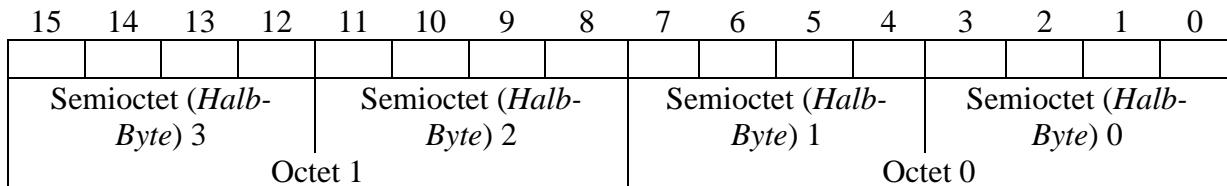


Figura 17. Un cuvânt IBM-PC (Einen IBM-PC Wort)

Evident, pentru cuvântul din **Figura 17**, semioctetul 0 și octetul 0 sunt respectiv semioctetul low și octetul low din cadrul locației. Similar, semioctetul 3 și octetul 1 sunt respectiv semioctetul high și octetul high din cadrul locației.

Offensichtlich sind für das Wort in Figura 17 das Halb-Byte 0 und das Byte 0 das Low Halb-Byte und das Low Byte innerhalb den Speicherort sind. In ähnlicher Weise das Halb-Byte 3 und das Byte 1 das High Halb-Byte und das High Byte innerhalb den Speicherort sind.

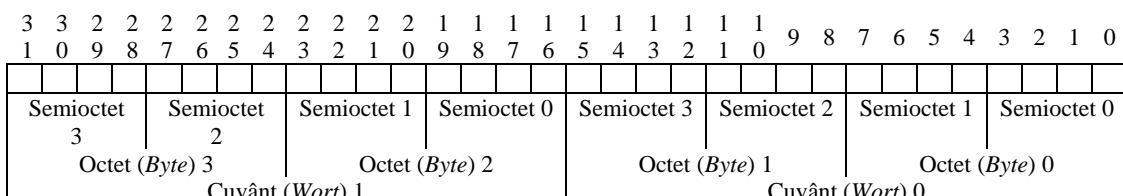


Figura 18. Un dublucuvânt IBM-PC (Einen IBM-PC Doppelwort)

Pentru dublucuvântul din **Figura 18**, semioctetul 0, octetul 0 și cuvântul 0 sunt respectiv semioctetul low, octetul low și cuvântul low din cadrul locației. Similar, semioctetul 7, octetul 3 și cuvântul 1 sunt respectiv semioctetul high, octetul high și cuvântul high din cadrul locației.

Für das Doppelwort in Figura 18 sind der Halb-Byte 0, das Byte 0 und das Wort 0 jeweils das Low (niedrigstwertige) Halb-Byte, das Low Byte und das Low Wort an der Speicherorts. Ebenso sind der Halb-Byte 7, das Byte 3 und das Wort 1 jeweils das High (höchstwertige) Halb-Byte, das High

Byte und das High Wort an der Speicherorts.

3.3 Tipuri elementare de date: dimensiuni ale standardelor de reprezentare

Cele mai utilizate tipuri elementare de date sunt caracterele, numerele întregi și numerele reale. Pentru fiecare dintre aceste tipuri de date sunt stabilite o serie de standarde de reprezentare a datei în memorie. În particular, standardul de reprezentare fixează dimensiunea celulei de memorie (*memory location*) pentru fiecare tip de dată.

Câteva dimensiuni ale locațiilor de reprezentare:

- Tip de date caracter.
 - 1 octet - standardul ASCII;
 - 2 octeți-standardul UNICODE.
- Tip de date întreg: 1, 2, 4, 8 octeți, depinde de sistemul de calcul.
- Tip de date real:
 - 4 octeți – standardul IEEE simplă precizie;
 - 8 octeți – standardul IEEE dublă precizie;
 - 6 octeți – standardul Turbo Pascal.

Câteva exemple de reprezentări ale unor date elementare. Conținuturile locațiilor le vom scrie în hexazecimal:

- Caracterele „M” și „m” se reprezintă:
 - 4D, respectiv 6D – standardul ASCII;
 - 004D, respectiv 006D – standardul UNICODE.
- Numerele 1 și -1, interpretate ca întregi pe 2 octeți se reprezintă 0001 respectiv FFFF (deoarece se folosește sistemul de numerație Complementul lui 2). Aceleași numere interpretate ca întregi pe 4 octeți se reprezintă 00000001 respectiv FFFFFFFF;
- Numerele 2 și -2, interpretate ca întregi pe 2 octeți se reprezintă 0002 respectiv FFFE. Aceleași numere interpretate ca

3.3 Elementare Datentypen: Dimensionen von die Darstellungsstandarden

Die am häufigsten verwendeten elementaren Datentypen sind Zeichen, Ganzzahlen und Realer Zahlen. Für jeden dieser Datentypen ist eine Reihe von Standards für die Datendarstellung im Speicher festgelegt. Insbesondere legt der Darstellungsstandard die Größe des Speicherplatzes (*memory location*) für jeden Datentyp fest.

Verschiedene Dimensionen von Darstellungsorten:

- Zeichen Datentyp.
 - 1 Byte – ASCII Standard;
 - 2-Byten – UNICODE Standard.
 - Ganzzahlen Datentyp: 1, 2, 4, 8 Byten, abhängig vom Computersystem.
 - Realer Datentyp:
 - 4 Byten – IEEE *einfache Präzision* Standard;
 - 8 Bytes – IEEE *doppelte Präzision* Standard;
 - 6 Bytes – der Turbo Pascal Standard.
- Einige Beispiele für die Darstellung von grundlegende Daten. Wir werden den Inhalt den Speicherorten in hexadezimal schreiben:
- Die Zeichen ”M” und ”m” sind:
 - 4D oder 6D – der ASCII-Standard;
 - 004D und 006D – der UNICODE-Standard.
 - Zahlen 1 und -1, interpretiert als 2-Byten-Ganzzahlen, werden durch 0001 bzw. FFFF dargestellt (da das Nummerierungssystem von Zweier Complement verwendet wird). Dieselben Zahlen, die als 4-Byten-Ganzzahlen interpretiert werden, sind 00000001 bzw. FFFFFFFF;
 - Zahlen 2 und -2, die als 2-Byten-Ganzzahlen interpretiert werden, werden durch 0002 bzw. FFFE dargestellt.

întregi pe 4 octeți se reprezintă 00000002 respectiv FFFFFFFE.

- Numerele 1 și -1, dar de această dată interpretate ca numere reale se reprezintă:
 - 3F800000, respectiv BF800000 – standardul IEEE simplă precizie,
 - 3FF000000000000, respectiv BFF000000000000 – standardul IEEE dublă precizie
 - 0000000081, respectiv 80000000081 – standardul Turbo Pascal.

Conform cu cele prezentate la numerotarea octetilor într-o locație, pentru exemplele de mai sus, avem:

- Octetul 0 al reprezentării caracterului „m” în standardul UNICODE are valoarea 6D;
- Octetul 1 al reprezentării caracterului „m” în standardul UNICODE are valoarea 00;
- Octetul 0 al reprezentării numărului -1 în standardul Turbo Pascal are valoarea 81;
- Octetul 5 al reprezentării numărului -1 în standardul Turbo Pascal are valoarea 80;
- Octetul 0 al reprezentării numărului -1 în standardul IEEE dublă precizie are valoarea 00;
- Octetul 6 al reprezentării numărului -1 în standardul IEEE dublă precizie are valoarea F0;
- Octetul 7 al reprezentării numărului -1 în standardul IEEE dublă precizie are valoarea BF.

3.4 Ordinea octetilor într-o locație; mașini little-endian și mașini big-endian

La nivelul unei locații, privită ca o entitate de sine stătătoare cu conținut interpretat în funcție de standardul de reprezentare, se disting două abstracții:

Dieselben Zahlen, die als 4-Byte-Ganzzahl interpretiert werden, sind 00000002 bzw. FFFFFFFE.

- Die Zahlen 1 und -1, aber diesmal jedoch als reelle Zahlen interpretiert, sind:
 - 3F800000 bzw. BF800000 – in der einfache Präzision IEEE Standard;
 - 3FF000000000000 bzw. BFF000000000000 – in der doppelte Präzision IEEE-Standard;
 - 0000000081 bzw. 80000000081 – in der Turbo Pascal Standard.

Entsprechend den in der Nummerierung der Bytes an einer Speicherort gezeigten haben wir für die obigen Beispiele:

- Das Byte 0 der Zeichendarstellung „m“ im UNICODE-Standard ist 6D;
- Das Byte 1 der Zeichendarstellung „m“ im UNICODE-Standard ist 00;
- Das Byte 0 der -1-Darstellung im Turbo Pascal-Standard ist 81;
- Das Byte 5 der -1-Darstellung im Turbo Pascal-Standard ist 80;
- Das Byte 0 der -1-Darstellung im IEEE-Standard mit doppelte Präzision ist 00;
- Das Byte Stelle der -1-Darstellung im IEEE-Standard mit doppelte Präzision ist F0;
- Das Byte 7 der -1-Darstellung im IEEE-Standard mit doppelter Präzision ist BF.

3.4 Die Reihenfolge der Bytes an einem Speicherort; little-endian und big-endian Maschinen

Auf der Ebene eines Speicherorts, der als eigenständige Entität mit interpretiertem Inhalt gemäß dem Repräsentationsstandard betrachtet wird, werden zwei Abstraktionen unterschieden:

- Numerotarea octetilor în cadrul unei locații fixată de standardul de reprezentare;
- Adresele octetilor care compun locația.

Așa cum am precizat deja, numerotările de conținuturi se fac de la dreapta spre stânga, iar adresele cresc de la stânga spre dreapta. În mod normal, se pune problema unei corespondențe între aceste două elemente.

Pe de o parte vorbim de *repräsentarea structurală* a unui tip de date, impusă de standardul de reprezentare. Pe de altă parte vorbim de *memorarea concretă* a datei în locație: în care octet al locației memorăm octetul 0 al reprezentării, în care octet al locației memorăm octetul 1 și.m.d.

Cu alte cuvinte trebuie stabilită o *corespondență* între ordinea octetilor impusă de reprezentarea structurală și adresele din locație în care se memorează valorile acestor octeți.

Această corespondență constituie o caracteristică a sistemului de calcul și ea poate fi una dintre următoarele două:

- Plasarea *little-endian*, în care octetul cu cea mai mică adresă din locație va conține octetul cu numărul 0 al reprezentării, octetul cu adresa următoare va conține octetul 1 al reprezentării și.m.d. (octetul „end” al reprezentării are adresa cea mai mică, „little”);
- Plasarea *big-endian*, în care octetul cu cea mai mare adresă din locație va conține octetul 0 al reprezentării, octetul cu adresa precedentă va conține octetul 1 al reprezentării și.m.d. (octetul „end” al reprezentării are adresa cea mai mare, „big”).

Spre exemplu, urmărim să reprezentăm numărul $(1025)_{10}$ într-o locație de patru

- Nummerierung der Bytes in einer durch den Darstellungsstandard festgelegten Speicherort;
- Adressen von Bytes, die der Speicherort bilden.

Wie bereits erwähnt, werden die Inhalte von rechts nach links nummeriert und die Adressen von links nach rechts erhöht. Normalerweise besteht eine Frage der Übereinstimmung zwischen diesen beiden Elementen.

Einerseits geht es um die *strukturelle Repräsentation* eines Datentyps, der durch den Repräsentationsstandard vorgegeben ist. Andererseits geht es um die *tatsächliche Speicherung* des Datums am Speicherort: in welchem Byte des Speicherorts speichern wir das Byte 0 der Darstellung, in welchem Byte des Speicherorts speichern wir Byte 1 usw.

Mit anderen Worten eine *Korrelation* zwischen der Reihenfolge des Byten, die die Strukturdarstellung auferlegen und die Adressen an dem Speicherort, an dem die Werte dieser Bytes gespeichert werden sollen.

Diese Entsprechung ist ein Merkmal des Computersystems und kann eine der folgenden zwei sein:

- *Little-Endian-Platzierung*: Das Byte mit der kleinsten Adresse am Speicherort enthält das 0-Byte der Darstellung, das Byte mit der nächsten Adresse enthält das Byte 1 der Darstellung usw. (Das „End“ -Byte der Darstellung hat die kleinste Adresse, „little“);
- *Big-Endian-Platzierung*, bei der das Byte mit der größten Adresse am Speicherort enthält das 0-Byte der Darstellung, das Byte mit der vorherigen Adresse enthält das Byte 1 der Darstellung usw. (Das „End“ -Byte der Darstellung hat die größte Adresse, „big“).

Zum Beispiel versuchen wir, die Zahl $(1025)_{10}$ an einer 4-Byte-Stelle darzustellen.

octeți. Pentru această dimensiune a locației, reprezentările lui în bazele 16 și 2 sunt $(00000401)_{16}$, respectiv $(00000000\ 00000000\ 00000100\ 00000001)_2$. Să presupunem că **B** este adresa locației în care numărul este plasat în ordinea big-endian, iar **L** este adresa locației în care același număr este plasat în ordinea little-endian. Cele două plasări sunt ilustrate în **Figura 19**, cu conținuturile octetilor scrise atât în hexazecimal, cât și în binar:

Für diese Dimension des Speicherorts sind seine Darstellungen in den Basen 16 und 2 $(00000401)_{16}$ bzw. $(00000000\ 00000000\ 00000100\ 00000001)_2$. Angenommen, **B** ist die Adresse des Speicherortes, an dem die Zahl in der Big-Endian-Reihenfolge platziert ist, und **L** ist die Adresse des Speicherortes, an dem die gleiche Zahl in der Little-Endian-Reihenfolge platziert ist. Die beiden Platzierungen sind in Figur 19 mit hexadezimalen und binären Byte-Inhalt dargestellt:

<i>Big-endian</i>	00 00000000	00 00000000	04 00000100	01 00000001
	Adresa (Adresse)	Adresa (Adresse)	Adresa (Adresse)	Adresa (Adresse)
	B	B+1	B+2	B+3
<i>Little-endian</i>	01 00000001	04 00000100	00 00000000	00 00000000
	Adresa (Adresse)	Adresa (Adresse)	Adresa (Adresse)	Adresa (Adresse)
	L	L+1	L+2	L+3

Figura 19. Plasările big-endian și little-endian (Die big-endian und little-endian Platzierungen)

Plasarea little-endian sau big-endian este o caracteristică a sistemului de calcul. De multe ori se folosește termenul de arhitectură *big-endian*, respectiv *arhitectură little-endian*. Un procesor are instrucțiuni specializate care să opereze cu fiecare tip de locație, instrucțiuni care „știu” standardul de reprezentare și ordinea de plasare. Utilizatorul trebuie doar să comande procesorului operația dorită și adresa locației de unde aceasta să își ia reprezentarea datelor.

Fiecare dintre cele două moduri de plasare are avantaje și dezavantaje. Prezentăm doar două criterii de comparare, fiecare dintre ele fiind avantajos pentru o arhitectură și dezavantajos pentru cealaltă. Aceste criterii sunt:

- Conversia unui număr întreg de la o reprezentare mai mare la una mai mică. De exemplu, dacă se cere ca un întreg

Little-Endian- oder Big-Endian-Platzierung ist ein Merkmal des Computersystems. Oft wird der Begriff „Big-Endian-Architektur“ oder „Little-Endian-Architektur“ verwendet. Ein Prozessor hat spezielle Anleitungen, um mit jeder Art von Speicherort zu arbeiten, Anleitungen, die den Repräsentationsstandard und die Reihenfolge der Platzierung „kennen“. Der Benutzer muss dem Prozessor nur die gewünschte Operation und die Adresse des Speicherortes befehlen, von dem er seine Datendarstellung bezieht.

Jede der beiden Platzierungen hat Vor- und Nachteile. Wir stellen nur zwei Vergleichskriterien vor, von denen jedes für eine Architektur vorteilhaft und das andere nachteilig ist. Diese Kriterien sind:

- Das Konvertieren einer Ganzzahl von einer größeren in eine kleinere Darstellung. Wenn beispielsweise eine 4-

dintr-o locație de 4 octeți, dar care începe de fapt pe 2 octeți, să fie prelucrat, cu aceeași valoare, ca și când ar face parte dintr-o locație pe 2 octeți. Comparativ, acest criteriu reprezintă un avantaj little-endian, dezavantaj big-endian.

- Depistarea biților de semn. Este unanim acceptat faptul că semnul unui număr, întreg sau real, se reprezintă pe un bit, cu valoarea 0 pentru număr pozitiv și cu valoarea 1 pentru număr negativ. Indiferent de standardul de reprezentare, bitul de semn este bitul high al octetului high din reprezentare. Comparativ, acest criteriu reprezintă un avantaj big-endian, dezavantaj little-endian.

De ce? Să luăm ca exemplu reprezentările numărului 2 pe 4 octeți, atât în plasarea big-endian, cât și în plasarea little-endian, ilustrate în **Figura 20**.

Pentru conversie, în cazul unei mașini little endian adresa locației rămâne L, indiferent dacă aceasta are dimensiunea de 4 octeți, de 2 octeți, sau chiar de 1 octet: se modifică doar dimensiunea locației (deci locația care conține numărul 2 va avea aici aceeași adresă L, indiferent dacă numărul este reprezentat pe 1, 2 sau 4 octeți!). La mașina big-endian, adresele locațiilor trebuie modificate în funcție de dimensiunea acestora: B+2 pentru locația de 2 octeți (conținând octeții de adrese B+2 și B+3) și B+3 pentru locația de 1 octet. Deci în cazul big-endian procesorul trebuie să facă în plus calculul de adresă.

Byte-Speicherort insgesamt, die jedoch tatsächlich 2 Byte umfasst, mit demselben Wert verarbeitet werden muss, als wäre sie Teil einer 2-Byte-Position. Im Vergleich ist dieses Kriterium ein vorteilhaft für Little-Endian, aber ein Nachteil für Big-Endian.

- Vorzeichenbit erkennen. Es ist allgemein anerkannt, dass das Vorzeichen einer ganzen oder reellen Zahl durch ein Bit mit dem Wert 0 für die positive Zahl und dem Wert 1 für die negative Zahl dargestellt wird. Unabhängig vom Darstellungsstandard ist das Vorzeichenbit das High-Bit der High-Byte-Darstellung. Im Vergleich ist dieses Kriterium ein vorteilhaft für Big-Endian, aber ein Nachteil für Little-Endian.

Warum? Nehmen wir als Beispiel die Darstellungen von der Zahl 2 um 4 Bytes sowohl in der Big-Endian-Platzierung als auch in der Little-Endian-Platzierung (siehe Figur 20).

Für der Konvertierung bei einem Little-Endian-Rechner bleibt die Speicherortadresse L, unabhängig davon, ob sie 4 Byte, 2 Byte oder sogar 1 Byte groß ist: Nur die Größe des Speicherorts ändert sich (der Speicherort mit die Zahl 2 hat hier also die gleiche Adresse L, unabhängig davon, ob die Zahl durch 1, 2 oder 4 Bytes dargestellt wird!). Für den Big-Endian-Rechner müssen die Standortadressen entsprechend ihrer Größe geändert werden: B + 2 für den 2-Byte-Standort (mit B + 2- und B + 3-Addressbytes) und B + 3 für den 1-Byte-Speicherort. Im Big-Endian-Fall muss der Prozessor die Adressberechnung also weiter ausführen.

<i>Big-endian</i>	00 00000000	00 00000000	00 00000000	02 00000010
	Adresa (Adresse)	Adresa (Adresse)	Adresa (Adresse)	Adresa (Adresse)
	B	B+1	B+2	B+3
<i>Little-endian</i>	02 00000010	00 00000000	00 00000000	00 00000000
	Adresa (Adresse)	Adresa (Adresse)	Adresa (Adresse)	Adresa (Adresse)
	L	L+1	L+2	L+3

Figura 20. Reprezentarea lui 2 în cele două tipuri de plasări (Die Darstellung von 2 in den zwei Arten von Platzierungen)

Pentru bitul de semn, în cazul unei mașini big-endian adresa octetului cu bitul de semn coincide cu adresa locației. Cu notațiile din **Figura 20**, bitul de semn, la mașina big-endian, se află la adresa B, indiferent de faptul că se va prelucra o locație de 4 octeți, de 2 octeți sau de 1 octet. În cazul unei mașini little-endian, bitul de semn se află în ultimul octet al locației (cel cu cea mai mare adresă), așa că pentru a-1 obține procesorul trebuie să calculeze adresa acestui octet: ea este L pentru locația de 1 octet, L+1 pentru locația de 2 octeți și L+3 pentru cea de 4 octeți.

Utilizatorul poate să interpreteze în mod diferit conținutul aceleiași locații! De exemplu, poate să memoreze într-un sir de 4 octeți un număr întreg, după care să comande operarea asupra aceleiași arii de memorie prin patru instrucțiuni care utilizează 4 locații consecutive de tip caracter reprezentat pe octet. În astfel de situații, când la momente de timp diferite se interpretează diferit aceeași arie de memorie, trebuie să se țină cont de ordinea de plasare și de standardele de reprezentare ale datelor elementare.

Sistemele de calcul de dimensiuni mari, cum ar fi procesoarele SPARC sau MOTOROLA, mașinile RISC, ca și supercalculatoarele CDC-Cyber sau CRAY,

Für das Vorzeichenbit stimmt im Fall einer Big-Endian-Maschine die Adresse des Bytes mit dem Vorzeichenbit mit der Adresse des Speicherorts überein. Mit den Notationen in Figura 20 befindet sich das Vorzeichenbit auf der Big-Endian-Maschine auf B, unabhängig davon, ob eine 4-Byte-, 2-Byte- oder 1-Byte-Speicherort verarbeitet wird. Bei einer Little-Endian-Maschine befindet sich das Vorzeichenbit im letzten Byte der Speicherort (dem mit der größten Adresse). Um den Prozessor zu erhalten, muss er die Adresse dieses Bytes berechnen: Es ist L für die 1-Byte-Speicherort, L + 1 für die 2-Byte-Speicherort und L + 3 für die 4 Bytes-Speicherort.

Der Nutzer kann den Inhalt desselben Ortes unterschiedlich interpretieren!

Beispielsweise kann es eine Ganzzahl in einer Folge von 4 Bytes speichern und dann denselben Speicherbereich mit vier Anleitungen befehlen, die vier aufeinanderfolgende Zeichenspeichernorten verwenden, die durch Byte dargestellt werden. In solchen Situationen wenn zu verschiedenen Zeiten derselbe Speicherbereich unterschiedlich interpretiert wird, müssen die Platzierungsreihenfolge und die Darstellungstandard für die elementaren Daten berücksichtigt werden.

Großrechnersysteme wie SPARC- oder MOTOROLA-Prozessoren, RISC-Maschinen und CDC-Cyber- oder CRAY-Supercomputer verwenden eine Big-Endian-

folosesc arhitectura big-endian. Calculatoarele uzuale actuale, în particular cele de tip IBM-PC, procesoarele INTEL și DEC-Alpha folosesc arhitectură little-endian. De o factură aparte sunt calculatoarele din familia PowerPC, care sunt mașini big-endian, ele „înțelegând” ambele arhitecturi.

3.5 Unități de capacitate a memoriei

Prin capacitatea de memorare a unui sistem de calcul înțelegem numărul total de octeți ai unității de memorie.

În practică se folosesc o serie de multipli ai numărului de octeți. Spre deosebire de multiplii folosiți în activitatea cotidiană, unitățile multipli ale capacitații de memorare sunt exprimate sub formă de puteri ale lui 2, astfel:

1 Ko	Kilo-octet	= 2^{10} octeți
1 Mo	Mega-octet	= 2^{20} octeți
1 Go	Giga-octet	= 2^{30} octeți
1 To	Tera-octet	= 2^{40} octeți.

Architektur. Gegenwärtige Computer, insbesondere IBM-PC-, INTEL- und DEC-Alpha-Prozessoren, verwenden eine Little-Endian-Architektur. Einem besonderen Typen sind die PowerPC-Computer, bei denen es sich um Big-Endian-Maschinen handelt, die beide Architekturen „verstehen“.

3.5 Speicherkapazitätseinheiten

Unter der Speicherkapazität eines Computersystems versteht man die Gesamtzahl der Bytes der Speichereinheit.

In der Praxis wird eine Vielzahl von Bytes verwendet. Im Gegensatz zu den im Alltag verwendeten Vielfachen werden Mehrfachspeicherkapazitätseinheiten wie folgt als Zweierpotenzen ausgedrückt:

1 KB	Kilo-Byte	= 2^{10} Byte
1 MB	Mega-Byte	= 2^{20} Byte
1 GB	Giga-Byte	= 2^{30} Byte
1 TB	Terra-Byte	= 2^{40} Byte.

3.6 Codificarea caracterelor

Pentru codificarea caracterelor tipăribile în vederea prelucrării lor automate, s-au definit o serie standarde de reprezentare. Aceste standarde atribuie câte un număr întreg fiecărui caracter, iar valorile de codificare a caracterelor dintr-o anumită grupă respectă anumite condiții. Existența acestor condiții este benefică pentru prelucrarea automată a caracterelor.

Un prim sistem de codificare stabilit a fost EBCDIC (*Extended Binary Decimal Interchange Code*), care codifică un caracter pe un octet folosind numere întregi din intervalul [0,255]. În prezent, cel mai folosit sistem de codificare este ASCII (*American Standard Code for Information Interchange*). Standardul ASCII este un cod pe 7 biți, folosind numerele întregi din

3.6 Zeichenkodierung

Um die druckbaren Zeichen für ihre automatische Verarbeitung zu codieren, wurden eine Reihe von Darstellungsstandarden definiert. Diese Standarden weisen jedem Zeichen eine Ganzzahl zu, und die Zeichencodierungswerte in einer bestimmten Gruppe erfüllen bestimmte Bedingungen. Das Vorhandensein dieser Bedingungen ist für die automatische Zeichenverarbeitung von Vorteil.

Ein erstes festgelegtes Codierungssystem war EBCDIC (*Extended Binary Decimal Interchange Code*), das ein Zeichen in einem Byte mit Ganzzahlen im Bereich [0,255] codiert. Derzeit das gebräuchlichste Codierungssystem ist der *American Standard Code for Information Interchange* (ASCII). Der ASCII-Standard ist ein 7-Bit-Code, der Ganzzahlen im Bereich [0,127]

intervalul [0,127]. În ASCII este codificat un caracter pe un octet, iar bitul 7 (cel de-al 8-lea) este automat 0. Practic, toate calculatoarele actuale folosesc ASCII pentru codificarea caracterelor. Firma IBM a propus (și la propunere au aderat practic toate marile case de software și hardware), extinderea ASCII folosind și cel de-al 8-lea bit din octet, pentru codificarea unor caractere grafice speciale.

În contextul cerințelor de internaționalizare impuse de existența Internet, în ultimii 10 ani se impune din ce în ce mai mult standardul de codificare UNICODE. Acesta codifică un caracter pe doi octeți, pentru a se permite codificarea simbolurilor și a caracterelor folosite în scrierile majorității limbilor de pe planetă.

În cele ce urmează ne rezumăm la prezentarea standardului de codificare ASCII. Standardul ASCII împarte caracterele în următoarele cinci grupe:

1. Literele mici ale alfabetului a, b,..., z.
2. Literele mari ale alfabetului A, B,..., Z.
3. Cifrele zecimale 0, 1,... , 9.
4. O serie de caractere speciale: spațiul, virgula, punctul, +, -, _, \$, & și.a.m.d
5. Set de caractere funcționale care nu apar la tipărire / afișare, ci doar dirijează tipărirea / afișarea.

Caracterele funcționale apar în tabelele de definiție sub forma unor grupuri de litere, ca de exemplu CR, LF, TAB, FF, BEL, BS și.a.m.d. CR provoacă deplasarea dispozitivului de afișare (tipărire) la început de rând (*Carriage Return*). LF sau NL provoacă deplasarea dispozitivului cu un rând mai jos (*Line Feed* sau *New Line*), păstrându-se poziția în cadrul rândului. În funcție de sistem, pentru a separa două linii dintr-un text se folosește fie LF, fie

verwendet. In ASCII wird ein Zeichen in einem Byte codiert, und Bit 7 (das achte) ist automatisch 0. Nahezu alle aktuellen Computer verwenden ASCII zum Codieren von Zeichen. IBM hat vorgeschlagen (und praktisch alle großen Software- und Hardwarehäuser haben sich dem Vorschlag angeschlossen), ASCII mithilfe des achte Bit von das Byte zu erweitern, um spezielle Grafikzeichen zu codieren.

Im Kontext der Internationalisierungsanforderungen, die das Internet gestellt hat, in den letzten 10 Jahren muss der UNICODE codieren-Standard immer mehr werden. Es codiert ein Zeichen durch zwei Bytes, um die Symbole und Zeichen zu codieren, die zum Schreiben der meisten Sprachen auf dem Planeten verwendet werden.

Im Folgenden fassen wir die Darstellung des ASCII-Kodierungsstandards zu präsentieren. Der ASCII-Standard unterteilt die Zeichen in die folgenden fünf Gruppen:

1. Die Kleinbuchstaben des Alphabets a, b, ..., z.
2. Großbuchstaben des Alphabets A, B, ..., Z.
3. Dezimalzahlen 0, 1, ..., 9.
4. Eine Reihe von Sonderzeichen: Leerzeichen, Komma, Punkt, +, -, _, \$, & etc.
5. Satz von Funktionszeichen, die nicht auf dem Druck / Display erscheinen, sondern nur direkt auf dem Druck / Display.

Funktionszeichen werden in den Definitionstabellen als Buchstabengruppen angezeigt, z. B. CR, LF, TAB, FF, BEL, BS usw. CR bewirkt, dass sich das Anzeigegerät (Print) an den oberen Rand der Zeile bewegt (*Carriage Return*). LF oder NL bewirkt, dass sich das Gerät eine Zeile nach unten bewegt [Zeilenvorschub (*Line Feed*) oder Neue Zeile (*New Line*)], wobei die Position in der Zeile beibehalten wird. Abhängig vom System wird zum Trennen von zwei

succesiunea de caractere CR LF. TAB este caracterul de tabulare, deci provoacă avansul dispozitivului la poziția următorului stop de tabulare (de obicei peste 5-8 caractere). FF (*Form Feed*) provoacă trecerea la pagina (ecranul) următoare (următor). BEL provoacă emiterea unui semnal sonor, iar BS (*backspace*) provoacă deplasarea dispozitivului de afișare (tipărire) cu o poziție spre stânga, în vederea ștergerii (supraimprimării) ultimului caracter.

Condițiile de codificare pe care le respectă standardul ASCII sunt:

- Toate caracterele funcționale au codul mai mic decât codul caracterului spațiu;
- Codul caracterului spațiu este mai mic decât codurile celorlalte caractere tipăribile;
- Literele mici sunt codificate prin 26 numere consecutive, în ordine alfabetică;
- Literele mari sunt codificate prin 26 numere consecutive, în ordine alfabetică;
- Cifrele zecimale sunt codificate prin 10 numere consecutive, în ordinea valorilor.

Textzeilen entweder LF oder die CR LF-Zeichenfolge verwendet. TAB ist das Tabulatorzeichen, sodass das Gerät zur nächsten Tabulatorposition vorrückt (normalerweise mehr als 5 bis 8 Zeichen). Mit FF (*Form Feed*) wechseln Sie zur nächsten Seite oder Bildschirm. BEL gibt einen Piepton aus und BS [Rücktaste (*backspace*)] bewirkt, dass das Anzeigegerät (Drucken) eine Position nach links verschoben wird, um das letzte Zeichen zu löschen (überdrucken).

Die Kodierungsbedingungen, denen der ASCII-Standard entspricht, sind:

- Alle Funktionszeichen haben einen niedrigeren Code als der Leerzeichencode;
- Der Leerzeichencode ist kleiner als die anderen Zeichen des druckbaren Zeichens;
- Kleinbuchstaben werden durch 26 aufeinanderfolgende Zahlen in alphabetischer Reihenfolge codiert;
- Große Buchstaben werden durch 26 aufeinander folgende Zahlen in alphabetischer Reihenfolge codiert;
- Dezimalstellen werden durch 10 aufeinanderfolgende Zahlen in der Reihenfolge der Werte codiert.

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0 000	NUL	(null)	32	20 040	 	Space		64	40 100	@	Ø	96	60 140	`	`		
1	1 001	SOH	(start of heading)	33	21 041	!	!	!	65	41 101	A	A	97	61 141	a	a		
2	2 002	STX	(start of text)	34	22 042	"	"	"	66	42 102	B	B	98	62 142	b	b		
3	3 003	ETX	(end of text)	35	23 043	#	#	#	67	43 103	C	C	99	63 143	c	c		
4	4 004	EOT	(end of transmission)	36	24 044	$	\$	\$	68	44 104	D	D	100	64 144	d	d		
5	5 005	ENQ	(enquiry)	37	25 045	%	%	%	69	45 105	E	E	101	65 145	e	e		
6	6 006	ACK	(acknowledge)	38	26 046	&	&	&	70	46 106	F	F	102	66 146	f	f		
7	7 007	BEL	(bell)	39	27 047	'	'	'	71	47 107	G	G	103	67 147	g	g		
8	8 010	BS	(backspace)	40	28 050	(((72	48 110	H	H	104	68 150	h	h		
9	9 011	TAB	(horizontal tab)	41	29 051)))	73	49 111	I	I	105	69 151	i	i		
10	A 012	LF	(NL line feed, new line)	42	2A 052	*	*	*	74	4A 112	J	J	106	6A 152	j	j		
11	B 013	VT	(vertical tab)	43	2B 053	+	+	+	75	4B 113	K	K	107	6B 153	k	k		
12	C 014	FF	(NP form feed, new page)	44	2C 054	,	,	,	76	4C 114	L	L	108	6C 154	l	l		
13	D 015	CR	(carriage return)	45	2D 055	-	-	-	77	4D 115	M	M	109	6D 155	m	m		
14	E 016	SO	(shift out)	46	2E 056	.	.	.	78	4E 116	N	N	110	6E 156	n	n		
15	F 017	SI	(shift in)	47	2F 057	/	/	/	79	4F 117	O	O	111	6F 157	o	o		
16	10 020	DLE	(data link escape)	48	30 060	0	0	0	80	50 120	P	P	112	70 160	p	p		
17	11 021	DC1	(device control 1)	49	31 061	1	1	1	81	51 121	Q	Q	113	71 161	q	q		
18	12 022	DC2	(device control 2)	50	32 062	2	2	2	82	52 122	R	R	114	72 162	r	r		
19	13 023	DC3	(device control 3)	51	33 063	3	3	3	83	53 123	S	S	115	73 163	s	s		
20	14 024	DC4	(device control 4)	52	34 064	4	4	4	84	54 124	T	T	116	74 164	t	t		
21	15 025	NAK	(negative acknowledge)	53	35 065	5	5	5	85	55 125	U	U	117	75 165	u	u		
22	16 026	SYN	(synchronous idle)	54	36 066	6	6	6	86	56 126	V	V	118	76 166	v	v		
23	17 027	ETB	(end of trans. block)	55	37 067	7	7	7	87	57 127	W	W	119	77 167	w	w		
24	18 030	CAN	(cancel)	56	38 070	8	8	8	88	58 130	X	X	120	78 170	x	x		
25	19 031	EM	(end of medium)	57	39 071	9	9	9	89	59 131	Y	Y	121	79 171	y	y		
26	1A 032	SUB	(substitute)	58	3A 072	:	:	:	90	5A 132	Z	Z	122	7A 172	z	z		
27	1B 033	ESC	(escape)	59	3B 073	;	:	:	91	5B 133	[[123	7B 173	{	{		
28	1C 034	FS	(file separator)	60	3C 074	<	<	<	92	5C 134	\	\	124	7C 174	|			
29	1D 035	GS	(group separator)	61	3D 075	=	=	=	93	5D 135]]	125	7D 175	}	}		
30	1E 036	RS	(record separator)	62	3E 076	>	>	>	94	5E 136	^	^	126	7E 176	~	~		
31	1F 037	US	(unit separator)	63	3F 077	?	?	?	95	5F 137	_	_	127	7F 177		DEL		

Source: www.LookupTables.com

Figura 21. Codul ASCII (Die ASCII Code)

3.7 Înmulțiri și împărțiri

Înmulțirea și împărțirea pe n biți impun următoarele restricții de dimensiune a locațiilor:

Înmulțirea pe n biți presupune că ambiii factori sunt reprezentați pe câte n biți, iar produsul lor va fi reprezentat pe $2 \times n$ biți. În cazul convenției cu semn, factorii înmulțirii vor fi reprezentați în cod complementar. Produsul va fi reprezentat tot în cod complementar și va respecta regula semnelor.

Drept consecință imediată a acestei dimensionări, rezultă că operația de înmulțire nu provoacă depășire! Într-adevăr, în convenția fără semn cea mai mare valoare posibilă pe n biți este $2^n - 1$, pătratul acestei valori este $2^{2n} - 2^{n+1} + 1$; acest număr începe pe $2n$ biți. În convenția cu semn, numărul -2^{n-1} are valoarea

3.7 Multiplikationen und Divisionen

Durch Multiplikation und Division in n Bits werden die folgenden Einschränkungen der Speicherorten auferlegt:

Die Multiplikation auf n Bits impliziert, dass beide Faktoren durch n Bits dargestellt werden und ihr Produkt durch $2 \times n$ Bits dargestellt wird. Im Falle einer Vorzeichenkonvention werden Multiplikationsfaktoren in einem komplementären Code dargestellt. Das Produkt wird auch im komplementären Code dargestellt und folgt der Vorzeichenregel.

Als unmittelbare Folge dieser Dimensionierung ergibt sich, dass die Multiplikation kein Überschwingen verursacht! Wirklich ist in der vorzeichenlosen Konvention der höchstmögliche Wert für n Bits $2^n - 1$, das Quadrat dieses Werts ist $2^{2n} - 2^{n+1} + 1$; diese Zahl passt auf $2n$ Bits. In der

absolută cea mai mare, iar pătratul acestia este 2^{2n-2} , număr care se poate reprezenta pe $2n-1$ biți, deci în cod complementar acest număr se poate reprezenta pe $2n$ biți.

Împărțirea pe n biți (oarecum invers față de înmulțire), impune condiția ca deîmpărțitul să fie reprezentat pe $2 \times n$ biți, iar împărțitorul pe n biți. Operația furnizează două rezultate: câtul reprezentat pe n biți și restul reprezentat tot pe n biți. În cazul convenției cu semn, deîmpărțitul și împărțitorul se vor reprezenta în cod complementar. Atât câtul, cât și restul vor fi, de asemenea, reprezentate în cod complementar. Câtul împărțirii va respecta regula semnelor. Important! Restul împărțirii va fi, în valoare absolută, mai mic decât valoarea absolută a împărțitorului și va avea același semn ca deîmpărțitul! De exemplu, $-7 : 3$ dă câtul -2 și restul -1 , adică $-7 = (-2) \times 3 + (-1)$.

În comparație, teorema împărțirii cu rest din aritmetică spune că restul trebuie să fie un număr pozitiv, deci în aritmetică avem $(-7) = (-3) \times 3 + 2$, deci câtul este -3 și restul 2 !

Operația de împărțire semnalează eroare la împărțitor zero (*Divide by zero!*). În cazul neîncadrării în dimensiuni semnalează depășire! Spre exemplu, împărțirea fără semn $1000 : 3$ se poate, formal, efectua, deoarece deîmpărțitul se poate reprezenta pe 16 biți și împărțitorul se poate reprezenta pe 8 biți. La această operație va apărea depășire, deoarece câtul este 333 și nu se poate reprezenta pe 8 biți! Restul împărțirii este 1 și se poate reprezenta pe un octet. Pentru a evita o astfel de situație, programatorul poate decide să facă operația pe 16 biți, adică să reprezinte deîmpărțitul pe 32 de biți și împărțitorul pe 16 biți,

Vorzeichenkonvention hat die -2^{n-1} -Zahl den höchsten Absolutwert und ihr Quadrat ist 2^{2n-2} , die durch $2n-1$ -Bits dargestellt werden kann. In einem komplementären Code kann diese Zahl also durch $2n$ -Bits dargestellt werden.

Die Division auf n Bits (etwas umgekehrt zur Multiplikation) impliziert, dass der Dividend durch $2 \times n$ Bits und der Divisor auf n Bits dargestellt wird. Die Operation liefert zwei Ergebnisse: den Quotient, der auf n Bits dargestellt ist, und den Rest, der ebenfalls durch n Bits dargestellt ist. Im Falle einer Vorzeichenkonvention werden der Dividend und der Divisor in einem komplementären Code dargestellt. Sowohl der Quotient als auch der Rest werden auch in einem komplementären Code dargestellt. Der Quotient von die Division der Regeln von die Zeichen folgt. Wichtig! Der Rest der Division ist in absoluten Zahlen niedriger als der absolute Wert der Divisor und hat das gleiche Vorzeichen wie der Dividend! Beispiel: $-7:3$ gibt den Quotient -2 und den Rest -1 an, dh $-7 = (-2) \times 3 + (-1)$.

Im Vergleich dazu besagt der Divisionssatz mit dem Rest der Arithmetik, dass der Rest eine positive Zahl sein muss. In der Arithmetik haben wir also $(-7) = (-3) \times 3 + 2$, also wie der Quotient ist -3 und der Rest 2 !

Die Division Operation signalisiert einen Divisor-Null Fehler (*Divide by zero!*). Wenn es nicht in die Größen passt, deutet es auf ein Überschwingen hin! Beispielsweise kann eine ungeteilte $1000 : 3$ Division formal durchgeführt werden, da der Dividend durch 16 Bit und die Divisor durch 8 Bit dargestellt werden kann. Bei dieser Operation kommt es zu einem Überlauf, weil der Quotient 333 ist und nicht durch 8 Bits dargestellt werden kann! Der Rest der Division ist 1 und kann durch ein Byte dargestellt werden. Um eine solche Situation zu vermeiden, kann der Programmierer entscheiden, die 16-Bit-Operation

urmând să obțină câtul și restul pe câte 16 biți (333 începe pe 16 biți și astfel nu vom mai avea depășire).

Operațiile de înmulțire și împărțire fără semn

Înmulțirea fără semn a numerelor întregi reprezentate binar se desfășoară conform algoritmului cunoscut, doar că se folosește baza 2. Pentru comoditate, vom considera $n = 4$. Să considerăm factorii 11 și 13. Înmulțirea lor în baza 2 este:

$$\begin{array}{r}
 1\ 0\ 1\ 1 \times \\
 1\ 1\ 0\ 1 \\
 \hline
 1\ 0\ 1\ 1 \\
 0\ 0\ 0\ 0 \\
 1\ 0\ 1\ 1 \\
 \hline
 1\ 0\ 1\ 1 \\
 \hline
 1\ 0\ 0\ 0\ 1\ 1\ 1\ 1
 \end{array}$$

Observații

1. Înmulțirea generează o serie de produse parțiale care sunt adunate. Există câte un produs parțial pentru fiecare cifră a înmulțitorului.
2. Produsele parțiale sunt fie deînmulțitul, fie 0.
3. Produsele parțiale nenule pot fi adunate unul câte unul. Ele sunt obținute deplasând deînmulțitul spre stânga cu câte o poziție.
4. Înmulțirea a două numere de câte n biți generează un produs care ocupă $2 \times n$ biți (motiv pentru care s-a impus restricția de dimensiune amintită mai sus).

Împărțirea fără semn se efectuează, de asemenea, după regulile cunoscute ale aritmeticii. Să împărțim pe 147 la 11. Împărțirea în baza 2 este:

durchzuführen, dh der Dividend durch 32 Bit und der Divisor durch 16 Bit dargestellt, sowie der Quotient und der Rest durch 16 Bits (333 passt auf 16 Bits und wir werden keinen Überlauf haben).

Vorzeichenlose Multiplikations- und Divisionsoperationen

Eine vorzeichenlose Multiplikation von Ganzzahlen, die binär dargestellt sind, wird nach dem bekannten Algorithmus durchgeführt, wobei jedoch nur die Basis 2 verwendet wird. Der Einfachheit halber betrachten wir $n = 4$. Berücksichtigen Sie die Faktoren 11 und 13. Die Multiplikation in der Basis 2 ist:

Bemerkungen

1. Die Multiplikation erzeugt eine Reihe von Teilprodukten, die gesammelt werden. Für jede Ziffer des Multiplikators gibt es ein Teilprodukt.
2. Teilprodukte werden der Multiplikator oder 0 sein.
3. Nicht-Null-Teilprodukte können einzeln hinzugefügt werden. Sie erhalten sie, indem Sie den Multiplikator um eine Position nach links bewegen.
4. Die Multiplikation von zwei n -Bit-Zahlen erzeugt ein Produkt, das $2 \times n$ Bits belegt (weshalb die oben erwähnte Größenbeschränkung auferlegt wurde).

Eine Vorzeichenlose Division wird auch nach bekannten arithmetischen Regeln durchgeführt. Teilen wir 147 durch 11. Die Division in Basis 2 lautet:

$$\begin{array}{r}
 1\ 0\ 0\ 1\ 0\ 0\ 1\ 1 \mid 1\ 0\ 1\ 1 \\
 1\ 0\ 1\ 1 \\
 \hline
 = 1\ 1\ 1\ 0 \\
 1\ 0\ 1\ 1 \\
 = 0\ 1\ 1\ 1 \\
 0\ 0\ 0\ 0 \\
 = 1\ 1\ 1\ 1 \\
 1\ 0\ 1\ 1 \\
 = 1\ 0\ 0
 \end{array}$$

Într-adevăr, $147 \div 11$ dă cîtuș 13 și restul 4.

În ultimă instanță, împărțirea în baza 2 se reduce la o succesiune de scăderi succesive combinate cu operații de deplasare.

Tatsächlich ergibt $147 \div 11$ der Quotient 13 und der Rest 4.

Letztendlich wird die Division in Basis 2 auf eine Abfolge aufeinanderfolgender Verringerungen in Kombination mit Verschiebungsoperationen reduziert.

Observații

- 1) Toți operanții implicați în operații sunt reprezentați în cod complementar, în conformitate cu cerințele de dimensionare expuse mai sus;
- 2) Atât la înmulțirea cu semn cât și la împărțirea cu semn, se respectă regula semnelor;
- 3) Pentru operația de împărțire, restul este în modul mai mic decât modulul împărtitorului, iar semnul restului este același cu semnul deîmpărțitului.

Algoritmii de înmulțire și împărțire în cod complementar nu pot fi preluati direct de la cei fără semn, ca în cazul adunării și scăderii.

3.8 Conversia la o locație de alte dimensiuni

Până acum am presupus că operanții au lungimi fixe, aşa cum pretind regulile de derulare a operațiilor. Dar ce-i de făcut atunci când, spre exemplu, trebuie să se convertească un cod complementar pe 8 biți

Bemerkungen

- 1) Alle Operanden, die an Operationen beteiligt sind, werden in einem komplementär Code gemäß den oben angegebenen Bemaßungsanforderungen dargestellt;
- 2) Sowohl dem Vorzeichenmultiplikation als auch der Vorzeichendivision folgt die Vorzeichenregel;
- 3) Für die Division Operation befindet sich der Rest im unteren Modus als der Divisor, und das Restzeichen ist dasselbe wie das Dividend.

Multiplikations- und Divisionsalgorithmen in komplementären Code können nicht direkt aus dem Vorzeichenlosen Code entnommen werden, wie im Fall von Addition und Subtraktion.

3.8 An einen anderen Speicherort konvertieren

Bisher haben wir angenommen, dass die Operanden feste Längen haben, wie es die Betriebsregeln von Operationen vorschreiben. Aber was tun, wenn beispielsweise ein komplementärer 8-Bit-

la unul pe 16 biți? Sau dacă trebuie să reducem un număr reprezentat fără semn pe 16 biți la unul similar pe 8 biți?

De fapt este vorba despre patru operații:

- Extensia cu semn a unui cod complementar într-o locație mai mare;
- Extensia cu zero a unui număr fără semn într-o locație mai mare;
- Contrația cu semn a unui cod complementar într-o locație mai mică;
- Contrația de zero a unui număr fără semn într-o locație mai mică.

Regulile de conversie sunt foarte simple, extensia cu semn înseamnă că în spațiul suplimentar toți biții vor avea ca valoare valoarea bitului de semn al reprezentării care se convertește. Extensia cu zero înseamnă că în spațiul suplimentar toți biții vor avea valoarea zero. Iată câteva exemple:

8 biți	16 biți: Extensie cu semn (Vorzeichenerweiterung)	32 biți: Extensie cu semn (Vorzeichenerweiterung)	16 biți: Extensie cu zero (Null-Erweiterung)	32 biți: Extensie cu zero (Null-Erweiterung)
80 1000 0000	FF80 1111 1111 1000 0000	FFFFFF80 1111 1111 1111 1111 1111 1111 1000 0000	0080 0000 0000 1000 0000	00000080 0000 0000 0000 0000 0000 0000 1000 0000
28 0010 1000	0028 0000 0000 0010 1000	00000028 0000 0000 0000 0000 0000 0000 0010 1000	0028 0000 0000 0010 1000	00000028 0000 0000 0000 0000 0000 0000 0010 1000
9A 1001 1010	FF9A 1111 1111 1001 1010	FFFFFF9A 1111 1111 1111 1111 1111 1111 1001 1010	009A 0000 0000 1001 1010	0000009A 0000 0000 0000 0000 0000 0000 1001 1010
7F 0111 1111	007F 0000 0000 0111 1111	0000007F 0000 0000 0000 0000 0000 0000 0111 1111	007F 0000 0000 0111 1111	0000007F 0000 0000 0000 0000 0000 0000 0111 1111
-	1020 0001 0000 0010 0000	00001020 0000 0000 0000 0000 0001 0000 0010 0000	----	00001020 0000 0000 0000 0000 0001 0000 0010 0000
-	8088 1000 0000 1000 1000	FFFF8088 1111 1111 1111 1111 1000 0000 1000 1000	----	00008088 0000 0000 0000 0000 1000 0000 1000 1000

Operațiile de contrație nu se pot executa întotdeauna. Spre exemplu, într-o locație pe 16 biți există numărul -448 în baza 10, care

Code in einen 16-Bit-Code konvertiert werden soll? Oder sollten wir eine vorzeichenlose 16-Bit-Zahl auf eine ähnliche 8-Bit-Zahl reduzieren?

In der Tat geht es um vier Operationen:

- Vorzeichenerweiterung eines komplementären Codes an einem größeren Speicherort;
- Null-Erweiterung einer Vorzeichenlosen Zahl an einer größeren Speicherort;
- Vorzeichenkontraktion eines komplementären Codes an einem kleineren Speicherort;
- Null-Kontraktion einer Vorzeichenlosen Zahl an einen kleineren Speicherort.

Die Konvertierungsregeln sind sehr einfach. Die Vorzeichenerweiterung bedeutet, dass in dem zusätzlichen Raum alle Bits den Wert des Vorzeichenbits der zu konvertierenden Darstellung bewerten. Die Null-Erweiterung bedeutet, dass in dem zusätzlichen Raum alle Bits Null sind. Hier einige Beispiele:

Die Kontraktionsoperationen können nicht immer durchgeführt werden. Beispielsweise gibt es an einer 16-Bit-Stelle die Nummer -

În cod complementar se reprezintă FE40. Dorim să efectuăm o contracție la 8 biți. Eliminând pur și simplu primul octet, se obține 40, adică numărul 64 în baza 10! Avem, evident, o situație de depășire. Cu alte cuvinte, contracțiile (conversii prin îngustare) se pot executa numai dacă NU se provoacă pierderea de informație.

Contrația cu semn se poate face numai dacă toți biții care se elimină coincid cu bitul de semn, adică cu primul bit care rămâne. Pentru contrația fără semn, trebuie ca toți biții care se elimină să fie zero. Tabelul următor prezintă câteva exemple.

448 in der Basis 10, die im komplementären Code FE40 dargestellt ist. Wir möchten eine 8-Bit-Kontraktion durchführen. Durch einfaches Entfernen des ersten Bytes erhalten Sie 40, also die Zahl 64 in der Basis 10! Wir haben offensichtlich eine Situation der Überschwingen. Mit anderen Worten, Kontraktionen (einschränkende Konvertierungen) können nur ausgeführt werden, wenn KEINE Informationen verloren gehen.

Eine Vorzeichenkontraktion kann nur durchgeführt werden, wenn alle entfernten Bits mit dem Vorzeichenbit, dh dem ersten verbleibenden Bit, übereinstimmen. Für eine Vorzeichenloskontraktion müssen alle Bits, die gelöscht werden, Null sein. Die folgende Tabelle zeigt einige Beispiele.

16 biți	8 biți: contrație cu semn (Vorzeichenkontraktion)	8 biți: contrație cu zero (Null-Kontraktion)
FF80 1111 1111 1000 0000	80 1000 0000	Depășire! (<i>Überschwingen!</i>) Se pierd 8 biți de 1 (<i>8 Bits von 1 gehen verloren</i>)
0028 0000 0000 0010 1000	28 0010 1000	28 0010 1000
FF9A 1111 1111 1001 1010	9A 1001 1010	Depășire! (<i>Überschwingen!</i>) Se pierd 8 biți de 1 (<i>8 Bits von 1 gehen verloren</i>)
FE40 1111 1110 0100 0000	Depășire! (<i>Überschwingen!</i>) Se schimbă bitul de semn (<i>Ändern des Vorzeichenbits</i>)	Depășire! (<i>Überschwingen!</i>) Se pierd 8 biți de 1 (<i>8 Bits von 1 gehen verloren</i>)
0100 0000 0001 0000 0000	Depășire! (<i>Überschwingen!</i>) Se schimbă bitul de semn (<i>Ändern des Vorzeichenbits</i>)	Depășire! (<i>Überschwingen!</i>) Se pierde ultimul bit de 1 (<i>Das Letzte Bit von 1 geht verloren</i>)
0088 0000 0000 1000 1000	Depășire! (<i>Überschwingen!</i>) Se schimbă bitul de semn (<i>Ändern des Vorzeichenbits</i>)	88 1000 1000

Curs 3 și 4

Contents

1. Organizarea unui sistem de calcul	3
1. Organisation eines Computersystem.....	3
2. Unitatea centrală (<i>Central Processing Unit – CPU</i>).....	8
2. Zentraleinheit (<i>Central Processing Unit – CPU</i>)	8
2.1 Ceasul sistem.....	10
2.1 Systemuhr.....	10
2.2 Performanțele unui sistem de calcul	14
2.2 Die Leistung eines Computersystems.....	14
2.2.1 Viteza unui sistem de calcul	14
2.2.1 Die Geschwindigkeit eines Computersystems	14
2.2.2 Criterii de evaluare a performanțelor.....	15
2.2.2 Leistungsbewertungskriterien	15
2.3 „Dimensiunea“ unui microprocesor	16
2.3 Die „Größe“ eines Mikroprozessors.....	16
3. Memoria	16
3. Der Speicher	16
3.1 Memoria internă	19
3.1 Interner Speicher	19
3.2 Memoria externă / secundară	22
3.2 Externer / sekundärer Speicher	22
3.3 Ierarhia memoriei	25
3.3 Die Speicherhierarchie.....	25
3.3.1. Memoria cache	27
3.3.1 Der Cache-Speicher.....	27
4. Dispozitivele periferice	29
4. Die Peripheriegeräte.....	29
4.1 Magistrale – structuri de interconectare.....	30
4.1 Busse - Verbindungsstrukturen	30
4.2. Magistrale I/O interne și interfețele asociate	33
4.2. Interne E / A-Busse und die zugehörigen Schnittstellen	33
4.2.1 Magistrala ISA	33
4.2.1 ISA-Bus	33

4.2.2 Magistrala PCI (<i>Peripheral Component Interconnect</i>)	34
4.2.3 Magistrala AGP	35
4.2.3 Der AGP Bus	35
4.2.4 Magistrala PCI Express	35
4.3 Magistrale I/O externe și interfețele asociate	39
4.3 E/A Busse und zugehörige Schnittstellen	39
4.3.1 IDE / ATA (<i>Integrated Drive Electronics / Advanced Technology Attachment</i>)	39
4.3.1 IDE / ATA (<i>Integrated Drive Electronics / Advanced Technology Attachment</i>)	39
4.3.2 SCSI (<i>Small Computer System Interface</i>)	40
4.3.2 SCSI (<i>Small Computer System Interface</i>)	40
4.3.3 Interfață serială	41
4.3.3 Serielle Schnittstelle	41
4.3.4 Interfață paralelă	41
4.3.4 Parallele Schnittstelle	41
4.3.5 USB (Universal Serial Bus)	42
4.3.5 USB (Universal Serial Bus)	42
5. Arhitectura microprocesoarelor x86 (IA-32)	42
5. Architektur von x86-Mikroprozessoren (IA-32)	42
5.1 Structura microprocesorului	42
5.1 Aufbau des Mikroprozessors	42
5.1.1 Reștricții generali ai EU	43
5.1.1 EU-Allgemeine Register	43
5.1.2 Flagurile	45
5.1.2 Flaggen	45
5.1.3 Reștricții de adresă și calculul de adresă	47
5.1.3 Adressregister und Adressberechnung	47
5.2 Reprezentarea instrucțiunilor mașină	53
5.2 Darstellung von MaschinenAnweisungen	53
5.3 Adrese FAR și NEAR	55
5.3 FAR- und NEAR-Adressen	55
5.4 Calculul <i>offset</i> -ului unui operand. Moduri de adresare	56
5.4 Berechnung des <i>Offsets</i> eines Operanden. Adressierungsarten	56

1. Organizarea unui sistem de calcul

Un sistem de calcul (SC) este un dispozitiv care lucrează automat, sub controlul unui program memorat. El prelucrează date cu scopul de a produce niște rezultate. Aceste rezultate se obțin ca efect al procesării.

Funcțiile de bază ale unui SC sunt:

- procesarea de date;
- memorarea de date;
- transferul de informații;
- controlul tuturor componentelor SC.

Arhitectura unui sistem de calcul poate fi analizată la nivel:

- *structural* (componentele și căile de comunicare între acestea);
- *logic* (funcțiile fiecărei componente în cadrul structurii).

Structura unui sistem de calcul este:

- hardware - partea de echipamente:
 - o unitatea centrală de procesare (*Central Processing Unit – CPU*);
 - o memoria;
 - o dispozitivele periferice;
- software - partea de programe:
 - o soft de sistem (aplicații destinate sistemului de calcul și sistemului de operare);
 - o soft utilizator (restul aplicațiilor);
- *firmware* - partea de microprograme.

Hardware-ul și software-ul se prezintă sub forma unor niveluri (componente) ierarhice. Fiecare componentă inferioară ascunde detaliile de implementare față de componentă superioară imediată. O astfel de abordare este reflectarea **principiului abstractizării** sau al proiectării ierarhice. Printre nivelurile ierarhice ale hardware-ului se pot identifica: structuri din siliciu

1. Organisation eines Computersystem

Ein Computersystem (CS) ist ein Gerät, das automatisch unter der Steuerung eines gespeicherten Programms arbeitet. Es verarbeitet Daten, um Ergebnisse zu erzielen. Diese Ergebnisse werden als Verarbeitungseffekt erhalten.

Die Grundfunktionen eines CS sind:

- Datenverarbeitung;
- Datenspeicherung;
- Übermittlung von Informationen;
- Steuerung aller CS-Komponenten.

Die Architektur eines Computersystems kann auf folgender Ebene analysiert werden:

- *strukturell* (die Komponenten und Kommunikationswege zwischen ihnen);
- *logisch* (die Funktionen jeder Komponente innerhalb der Struktur).

Die Struktur eines Computersystems ist:

- Hardware - Teil der Ausrüstung:
 - o eine Zentraleinheit (*Central Processing Unit – CPU*);
 - o ein Speicher;
 - o Peripheriegeräte;
- Software – der Teil von Programmen:
 - o Systemsoftware (Anwendungen für das Computersystem und das Betriebssystem);
 - o ein Software-Benutzer (der Rest der Anwendungen);
- *firmware* – der Teil den Mikroprogrammen.

Die Hardware und Software besteht aus hierarchischen Ebenen (Komponenten). Jede untere Komponente verbirgt die Implementierungsdetails vor der unmittelbar oberen Komponente. Ein solcher Ansatz spiegelt **das Prinzip der Abstraktion** oder hierarchischen Gestaltung wider.

Unter den Hierarchieebenen der Hardware können wir identifizieren:

sau alte materiale (componente electronice (tranzistori etc.), componente logice, circuite logice, unități funcționale ale sistemului (ALU, CU etc.), componente ale calculatorului (CPU, memorie, sistem de I/O), iar nivelurile ierarhice ale software-ului sunt reprezentate de limbajul mașină, limbajul de asamblare și limbaje de nivel înalt.

Arhitectura (organizarea) unui sistem de calcul se referă la acele atrbute ale sistemului care sunt vizibile programatorului și care au un impact direct asupra execuției unui program: setul de instrucțiuni mașină, caracteristicile de reprezentare a datelor, modurile de adresare și sistemul de intrare / ieșire (I/O). Din punct de vedere organizatoric, componentele unui SC sunt:

- Unitatea de Control;
- Unitatea de Execuție, numită și Calea de Date;
- Memoria;
- Sistemul de Intrare (*input*) / ieșire (*output*) = sistemul de I/O sau de I/E;
- Structuri de interconectare a componentelor de mai sus (Magistrale), unde Unitatea de Control și Calea de date sunt componente ale procesorului.

Această organizare este independentă de tehnologia hardware adoptată pentru construcția sistemului de calcul. Orice componentă a unui SC poate fi încadrată în una din aceste 5 categorii.

Văzută dintr-un alt unghi, arhitectura unui SC este compusă din multimea instrucțiunilor mașină și organizarea mașinii.

Mulțimea instrucțiunilor mașină (*Instruction Set Architecture – ISA*) este o interfață cheie între nivelurile de abstractizare. Ea este interfața dintre hardware și software-ul de nivel scăzut. O astfel de interfață permite unor

Siliziumstrukturen oder andere Materialien (elektronische Komponenten (Transistoren usw.), Logikkomponenten, Logikschaltungen, Funktionseinheiten des Systems (ALU, CU usw.), Computerkomponenten (CPU, Speicher, E / A-System) und die Hierarchieebenen der Software werden durch Maschinensprache, Assemblersprache und übergeordnete Sprachen dargestellt.

Die Architektur (Organisation) eines Computersystems bezieht sich auf diejenigen Systemattribute, die für den Programmierer sichtbar sind und die direkten Einfluss auf die Ausführung eines Programms haben: den Maschinenbefehlssatz, die Datenrepräsentationseigenschaften, die Adressierungsmodi und das Eingabe- / Ausgabesystem. (I / O). Aus organisatorischer Sicht sind die Komponenten eines CS:

- Die Steuereinheit;
- Die Ausführungseinheit, auch Datenpfad genannt;
- Der Speicher;
- Der Eingabe- / Ausgabesystem = der E / A-System;
- Verbindungsstrukturen der oben genannten Komponenten (Magistrale), dabei sind die Steuereinheit und der Datenpfad Komponenten des Prozessors.

Diese Organisation ist unabhängig von der für den Aufbau des Computersystems verwendeten Hardwaretechnologie. Jede Komponente eines CS kann in eine dieser 5 Kategorien fallen.

Aus einem anderen Blickwinkel besteht die Architektur eines SC aus der Vielzahl von Maschinenbefehlen und der Organisation der Maschine.

Die Befehlssatzarchitektur (*Instruction Set Architecture, ISA*) ist eine wichtige Schnittstelle zwischen Abstraktionsebenen. Es ist die Schnittstelle zwischen niedriger Pegel Hardware und Software. Eine solche Schnittstelle ermöglicht es verschiedenen

implementări diferite ale SC să ruleze programe identice, caz în care vorbim despre calculatoare compatibile (de exemplu – „calculatoarele compatibile IBM PC” nu au același hardware, dar au același set de instrucțiuni).

ISA definește:

- organizarea SC, modul de stocare a informației (registre, memorie);
- tipurile și structurile de date (codificări, reprezentări);
- formatul instrucțiunilor;
- setul de instrucțiuni (codurile operațiilor) pe care micropresorul le poate efectua;
- modurile de adresare și de accesare a datelor și instrucțiunilor;
- condițiile de excepție;

Organizarea unei mașini se referă la:

- implementarea, capacitatea și performanța unităților funcționale;
- interconexiunile dintre aceste unități;
- fluxul de informație dintre unități;
- controlul fluxului de informație.

Majoritatea calculatoarelor actuale sunt construite pe baza arhitecturii von Neumann. Ideea de plecare este de a utiliza memoria internă pentru a stoca secvențe de control în vederea îndeplinirii unei anumite sarcini – secvențe de programe; astfel, putem vorbi despre mașini programabile. Aceasta reprezintă așa-numitul concept al **programului memorat** (*stored-program concept*). În contrast, primele mașini erau construite pentru îndeplinirea unei anumite operații și era necesară modificarea acestora pentru a putea efectua un alt tip de operație.

CS-Implementierungen, identische Programme auszuführen. In diesem Fall handelt es sich um kompatible Computer (z. B. "IBM PC-kompatible Computer" haben nicht dieselbe Hardware, aber denselben Befehlssatz).

ISA definiert:

- Organisation von der CS, Informationsspeichermodus (Register, Speicher);
 - Datentypen und -strukturen (Kodierung, Darstellungen);
 - Format der Anweisungen;
 - den Befehlssatz (die Operationscoden), den der Mikroprozessor ausführen kann;
 - Möglichkeiten der Adressierung und des Zugriffs auf Daten und Anweisungen;
 - außergewöhnliche Bedingungen;
- Die Organisation eines CS bezieht sich auf:
- Implementierung, Kapazität und Leistung von Funktionseinheiten;
 - die Verbindungen zwischen diesen Einheiten;
 - Informationsfluss zwischen Einheiten;
 - Kontrolle des Informationsflusses.

Die meisten aktuellen Computer basieren auf der von Neumann-Architektur. Die Idee des Startens besteht darin, den internen Speicher zum Speichern von Steuersequenzen zu verwenden, um eine bestimmte Aufgabe auszuführen - Programmsequenzen; Somit können wir über programmierbare Maschinen sprechen. Dies ist das sogenannte **Speicherprogrammkonzept** (*stored-program concept*). Im Gegensatz dazu wurden die ersten Maschinen gebaut, um eine bestimmte Operation auszuführen, und es war notwendig, sie zu modifizieren, um eine andere Art von Operation auszuführen.

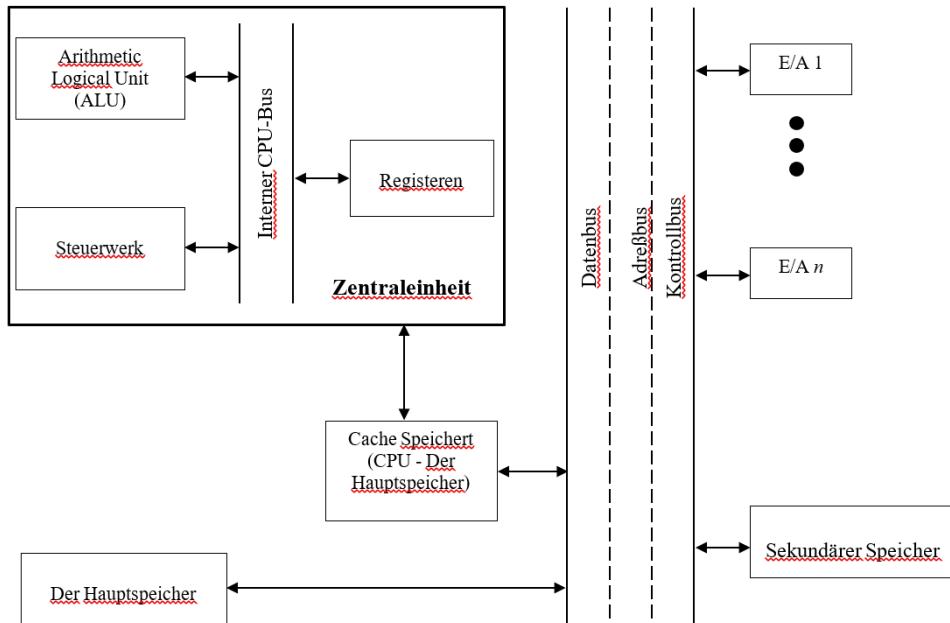


Figura (Abbildung) 1. Arhitectura unui sistem de calcul (Die Architektur eines Computersystems)

Caracteristicile arhitecturii von Neumann:

- atât datele, cât și instrucțiunile sunt reprezentate ca siruri de biți și sunt stocate într-o memorie *read-write*. Nu se poate face diferență între date și instrucțiuni prin simpla citire a unei locații de memorie – trebuie cunoscut ce reprezintă pentru a-i se stabili semnificația (de exemplu: valoarea 98h – poate reprezenta o valoare a unei variabile de dimensiune egală cu un cuvânt sau codul instrucțiunii mașinăcorespunzător instrucțiunii „8086 CBW” din limbajul de asamblare);
- conținutul memoriei se poate accesa în funcție de locație (adresă), indiferent de tipul informației conținute;
- execuția unui set de instrucțiuni se efectuează secvențial, prin citirea de instrucțiuni consecutive din memorie.

O altă perspectivă este oferită de Figura 2, în care se pot vedea etapele execuției unui program într-un sistem de calcul construit pe baza paradigmelor von Neumann. Arhitectura este reprezentată într-o formă simplificată.

Merkmale der von Neumann-Architektur:

- Sowohl Daten als auch Anweisungen werden als Zeichenfolgen von Bits dargestellt und in einem Lese- / Schreibspeicher gespeichert. Es ist nicht möglich, zwischen Daten und Anweisungen durch einfaches Lesen eines Speicherorts zu unterscheiden – es muss bekannt sein, was es darstellt, um seine Bedeutung zu bestimmen (zum Beispiel: der Wert 98h – es kann einen Wert einer Variablen darstellen, die die gleiche Größe wie ein Wort oder der Maschinenanweisungscode hat, der der Anweisung "8086 CBW" in der Assemblersprache entspricht);
 - Auf den Inhalt des Speichers kann in Abhängigkeit von der Position (Adresse) unabhängig von der Art der enthaltenen Informationen zugegriffen werden.
 - Die Ausführung eines Befehlssatzes wird sequentiell ausgeführt, indem aufeinanderfolgende Anweisungen aus dem Speicher gelesen werden.
- Eine andere Perspektive bietet Abbildung 2, in der man die Schritte der Ausführung eines Programms in einem Computersystem nach dem von Neumann-Muster sehen können. Die Architektur ist vereinfacht dargestellt.

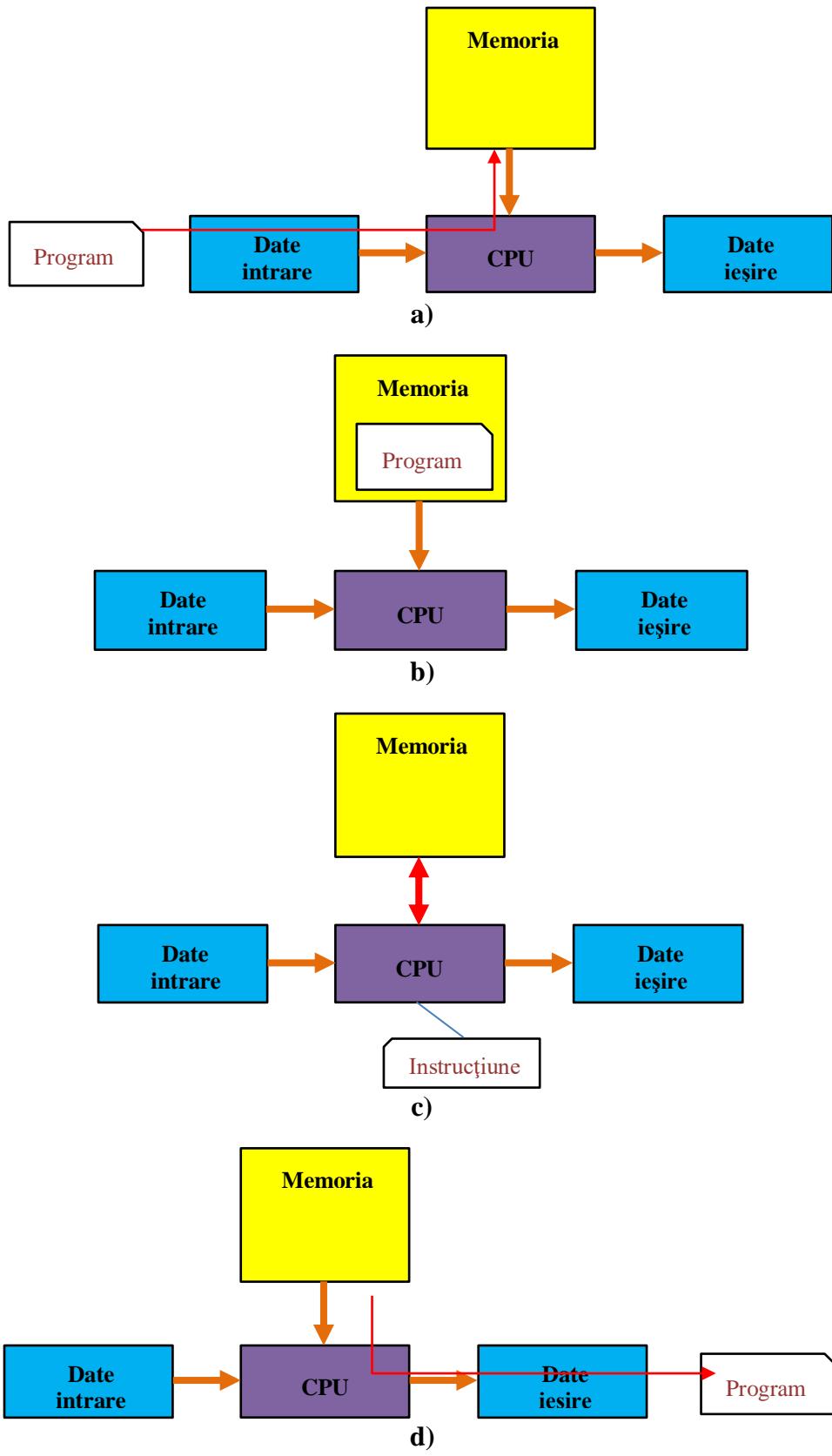


Figura 2. Etapele execuției unui program (Die Phasen der Ausführung eines Programms)

2. Unitatea centrală (Central Processing Unit – CPU)

Procesorul controlează modul de operare a calculatorului și execută funcțiile de procesare a datelor.

Funcțiile unui CPU sunt:

- obținerea instrucțiunilor care trebuie executate;
- obținerea datelor necesare instrucțiunilor;
- procesarea datelor (execuția instrucțiunilor);
- furnizarea rezultatelor obținute.

Putem identifica două componente de bază la nivelul unui CPU:

- Unitatea Aritmetică-Logică (*Arithmetic Logic Unit – ALU*);
- Unitatea de Comandă și Control (*Control Unit – CU*);

iar pentru îndeplinirea funcțiilor de mai sus, CPU mai are nevoie de:

- registre – acestea sunt dispozitive de stocare temporară a datelor și informațiilor de control (instrucțiunile), de capacitate mică și viteză de acces mare;
- magistrale interne CPU – dispozitive pentru comunicare între componente ale CPU și comunicare cu exteriorul.

Unitatea Aritmetică-Logică execută operații aritmetice și logice asupra datelor.

Unitatea de Comandă și Control este componenta CPU care dirijează toate componentele sistemului de calcul prin controlul direct asupra acestora. Pentru aceasta, trimite semnale de control în interiorul CPU și către magistrala sistem, sau captează semnale dinspre magistrala sistem. Nu execută instrucțiuni, ci le decodifică și comandă altor componente execuția efectiva a acestora. Regulile după care funcționează CU sunt codificate în *Programmable Logic Array* (PLA –

2. Zentraleinheit (Central Processing Unit – CPU)

Der Prozessor steuert die Funktionsweise des Computers und führt Datenverarbeitungsfunktionen aus.

Die Funktionen einer CPU sind:

- Einholen der auszuführenden Anweisungen;
- Einholen der für die Anweisungen erforderlichen Daten;
- Datenverarbeitung (Ausführung von Anweisungen);
- Bereitstellung der erzielten Ergebnisse.

Wir können zwei grundlegende Komponenten einer CPU identifizieren:

- Arithmetisch-Logische Einheit (*Arithmetic Logic Unit – ALU*);
- Befehls- und Steuereinheit (*Control Unit – CU*);

und um die obigen Funktionen auszuführen, benötigt die CPU:

- Registern – Dies sind Geräte zur vorübergehenden Speicherung von Steuerdaten und -informationen (Anweisungen) mit geringer Kapazität und hoher Zugriffsgeschwindigkeit.
- CPU-interne Busse – Geräte für die Kommunikation zwischen CPU-Komponenten und die Kommunikation nach außen.

Die *Arithmetic-Logic-Einheit* führt arithmetische und logische Operationen an den Daten aus.

Die Steuer- und Befehlseinheit ist die CPU-Komponente, die alle Komponenten des Computersystems durch direkte Steuerung steuert. Dazu sendet es Steuersignale innerhalb der CPU und an den Systembus oder empfängt Signale vom Systembus. Es führt keine Anweisungen aus, sondern decodiert und befiehlt anderen Komponenten, diese effektiv auszuführen. Die Regeln, nach denen CU arbeitet, sind im *Programmable Logic Array* (PLA – ein programmierbares Gerät, das für die

dispozitiv programabil utilizat pentru implementarea de circuite logice combinaționale AND / OR) sau într-o memorie de tip ROM (*Read-Only Memory*). Pentru execuția unei instrucțiuni, aceasta și datele necesare trebuie aduse din memoria secundară sau de la un dispozitiv de intrare în memoria principală. CU coordonează *ciclul de execuție a instrucțiunii*:

- citirea instrucțiunii din memoria principală (FETCH);
- decodificarea acesteia (DECODE - tipul instrucțiunii, numărul de argumente etc.);
- obținerea operanzilor necesari instrucțiunii (READ MEMORY);
- execuția ei (EXECUTE; dacă e cazul, ALU primește controlul pentru efectuarea operației aritmetice sau logice implicate);
- furnizarea rezultatelor în registre sau în memorie (STORE).

Apoi, CU poate iniția transferul acestor date rezultate din memoria internă către un dispozitiv de ieșire sau către un dispozitiv de stocare secundar.

Cunoșcând structura unui *CPU* și pașii din ciclul de execuție a unei instrucțiuni, putem contura structura funcțională a procesorului: *modulul de control și calea de date (Datapath)*.

Implementierung von UND / ODER-Verknüpfungslogikschaltungen verwendet wird) oder in einem ROM (Nur-Lese-Speicher, *Read-Only Memory*) codiert. Zur Ausführung einer Anweisung müssen dieser und die erforderlichen Daten aus dem Sekundärspeicher oder von einem *Eingabegerät* in den Hauptspeicher gebracht werden. Der CU koordiniert den Ausführungszyklus der Anweisung:

- Lesen die Anweisung aus dem Hauptspeicher (FETCH);
- Decodierung (DECODE – Art der Anweisung, Anzahl der Argumente usw.);
- Erhalten der für den Befehl benötigten Operanden (READ MEMORY);
- seine Ausführung (EXECUTE; ALU erhält gegebenenfalls die Steuerung zur Durchführung der betreffenden arithmetischen oder logischen Operation);
- Bereitstellen von Ergebnissen in Registern oder im Speicher (STORE).

Dann kann die CU die Übertragung dieser Daten vom internen Speicher zu einem Ausgabegerät oder einem sekundären Speichergerät initiieren.

Wenn wir die Struktur einer CPU und die Schritte im Ausführungszyklus einer Anweisung kennen, können wir die Funktionsstruktur des Prozessors skizzieren: *das Steuermodul* und *den Datenpfad (Datapath)*.

1	2	3	4	5
FETCH	DECODE	READ MEMORY	EXECUTE	STORE sau (oder) WRITE BACK
Adresa instrucțiunii (Adresse der Anweisung)	Decodificare instrucțiune (Dekodierung der Anweisung)	Citire operanzi din memorie (Liest Operanden aus dem Speicher)	Efectuare operație (UAL) (Operationsausführung - ALU)	Scriere rezultat (Schreiben das Ergebnis)
Copierea instrucțiunii într-un registru (Kopieren die Anweisung in einem Register)	Generare semnale pentru execuție (Signale zur Ausführung)			Scriere stare (Schreibstatus)

	generieren)			
<i>Uneori aceste 2 faze sunt prezentate combinate în una singură, EXECUTE (Manchmal werden diese beiden Phasen zu einer zusammengefasst)</i>				

Figura 3. Etapele execuției unei instrucțiuni (Die Schritte zum Ausführen einer Anweisung)

Modulul de control este răspunzător de comunicarea cu exteriorul CPU (preluarea și interpretarea instrucțiunilor și transmiterea rezultatelor), precum și de controlul execuției instrucțiunilor (emiterea semnalelor de control către calea de date și recepționarea semnalelor de stare dinspre aceasta).

Din **Calea de date** fac parte componente de stocare (registre de date), unități funcționale (ALU) și căi de comunicare. Practic, calea de date definește multimea operațiilor efectuate asupra datelor în vederea execuției unei instrucțiuni și drumul parcurs de datele prelucrate în decursul execuției operațiilor.

2.1 Ceasul sistem

Fiecare CPU are un ceas intern care produce și trimit semnale electrice pe magistrala de control pentru a sincroniza operațiile sistemului. Semnalele alternează valori 0 și 1 cu o anumită frecvență numită *frecvența ceasului sistem*. Timpul necesar trecerii din starea 0 în 1 și apoi înapoi în 0 se numește perioada ceasului sau ciclu de ceas (*clock cycle*).

Frecvența de ceas este de fapt numărul de cicluri de ceas pe secundă și este măsurată în hertzii.

Observație: $1 \text{ Hz} = 1 \text{ ciclu de ceas / secundă}$. Frecvența = $1 / \text{perioada}$

*Uneori aceste 2 faze sunt prezentate combinate în una singură, EXECUTE
(Manchmal werden diese beiden Phasen zu einer zusammengefasst)*

Das Steuermodul ist verantwortlich für die Kommunikation mit der externen des CPU (Entgegennahme und Interpretation der Anweisungen und Übertragung der Ergebnisse) sowie für die Steuerung der Ausführung der Anweisungen (Senden der Steuersignale an den Datenpfad und Empfangen der Statussignale von dieser).

Der Datenpfad enthält Speicherkomponenten (Datenregister), Funktionseinheiten (ALU) und Kommunikationspfade. Grundsätzlich definiert der Datenpfad den Satz von Operationen, die an den Daten ausgeführt werden, um eine Anweisung auszuführen, und den Pfad, den die während der Ausführung der Operationen verarbeiteten Daten zurücklegen.

2.1 Systemuhr

Jede CPU verfügt über eine interne Uhr, die elektrische Signale erzeugt und an den Steuerbus sendet, um den Systembetrieb zu synchronisieren. Die Signale wechseln sich mit 0 und 1 mit einer bestimmten Frequenz ab, die als Systemtaktfrequenz bezeichnet wird. Die Zeit, die benötigt wird, um vom Zustand 0 auf 1 und dann wieder auf 0 zu schalten, wird als Taktperiode oder Taktzyklus (*clock cycle*) bezeichnet.

Die Taktfrequenz ist tatsächlich die Anzahl der Taktzyklen pro Sekunde und wird in Hertz gemessen.

Hinweis: $1 \text{ Hz} = 1 \text{ Taktzyklus / Sekunde}$. Frequenz = $1 / \text{Periode}$

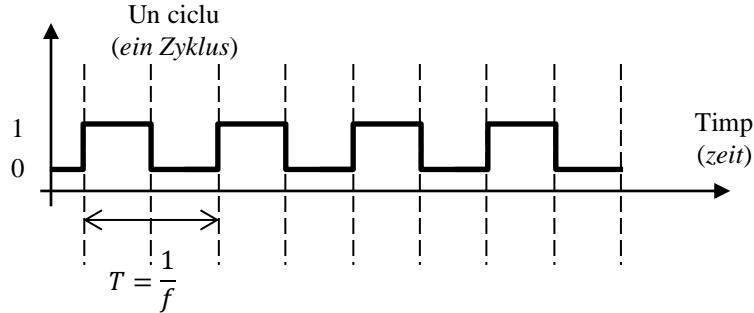


Figura 4. Semnalul periodic de tact sau clock (Periodisches Takt- oder clock Signal)

Toate operațiile efectuate de microprocesor sunt sincronizate cu ceasul sistem; aceasta înseamnă că procesorul nu poate efectua de o manieră secvențială operații mai rapid decât frecvența de tact a ceasului (a unității de timp a ceasului). Astfel, un procesor cu o viteză de ceas de 200 MHz are un ceas intern care generează 200.000.000 de impulsuri pe secundă. Un ciclu de ceas pentru un astfel de procesor are o durată de $1 / 200.000.000$ secunde.

Un ciclu de ceas este în general, pentru un procesor, unitatea de bază pentru măsurarea timpului. Este cea mai mică cantă de timp sesizabilă de către procesor. Toate activitățile și instrucțiunile executate de procesor sunt executate în general în multipli ai ciclului de ceas (din punct de vedere al duratei de execuție). Fiecare instrucțiune mașină necesită un număr fix, cunoscut (în general) de cicluri procesor. Există diferențe în durata execuției instrucțiunilor mașină în funcție de natura parametrilor și de mediul lor de stocare. Astfel o instrucțiune **MOV destinație, sursă** se execută între 2 și 14 cicluri de ceas procesor în funcție de natura argumentelor destinație și sursă care pot fi: registri generali, registri de segment, constante, adrese de memorie. Fiecare combinație posibilă a acestor tipuri de parametri va rezulta într-o durată de execuție specifică în cicluri de ceas procesor.

Alle vom Mikroprozessor ausgeführten Operationen werden mit der Systemuhr synchronisiert. Dies bedeutet, dass der Prozessor keine Operationen ausführen kann, die sequentiell schneller als die Taktfrequenz (Takteinheit der Zeit) sind. Ein Prozessor mit einer Taktrate von 200 MHz verfügt somit über einen internen Takt, der 200.000.000 Impulse pro Sekunde erzeugt. Ein Taktzyklus für einen solchen Prozessor hat eine Dauer von $1 / 200.000.000$ Sekunden.

Ein Taktzyklus ist für einen Prozessor in der Regel die Grundeinheit für die Zeitmessung. Dies ist die kürzeste Zeit, die der Prozessor erkennen kann. Alle vom Prozessor ausgeführten Aktivitäten und Anweisungen werden im Allgemeinen in Vielfachen des Taktzyklus (in Bezug auf die Ausführungszeit) ausgeführt. Jeder Maschinenbefehl erfordert eine feste Nummer, die (allgemein) aus den Prozessorzyklen bekannt ist. Die Ausführung der Maschinenbefehle unterscheidet sich je nach Art der Parameter und deren Speicherumgebung. Somit läuft eine **MOV Ziel, Quelle** Anweisung, zwischen 2 und 14 Prozessortaktzyklen, abhängig von der Art der Ziel- und Quellargumente, die sein können: allgemeine Register, Segmentregister, Konstanten, Speicheradressen. Jede mögliche Kombination dieser Parametertypen führt zu einer bestimmten Ausführungszeit in Prozessortaktzyklen.

Timpul de execuție al unei instrucțiuni măsurat în cicluri de ceas procesor poartă denumirea *Cycles per Instruction (CPI)*. Încă de la primele procesoare apărute pe piață, modul de execuție al instrucțiunilor mașină a fost unul *secvențial*. Procesorul preia o instrucțiune, petrece un număr de cicluri de ceas la execuția completă a acesteia, după care trece la următoarea instrucțiune și a.m.d.

În general CPI se referă la numărul *mediu* de cicluri procesor per instrucțiune executată de procesor – în raport cu toate instrucțiunile ce compun un program sau în raport cu întreg setul de instrucțiuni al procesorului.

Observație: Deși o instrucțiune **MOV** se execută în același număr N de cicluri procesor, durata de execuție în timp (secunde) este diferită de la un procesor la altul. Astfel un procesor ce rulează la viteza de 200 MHz va executa o instrucțiune **MOV** în 20 nanosecunde, în timp ce un procesor ce rulează la 1 GHz o va executa de aproximativ 5 ori mai rapid (4 nanosecunde).

Întrucât în zilele noastre s-a ajuns la o limită tehnologică din punct de vedere al frecvenței de ceas a procesoarelor, se urmărește creșterea vitezei acestora prin alte metode. Cea mai des întâlnită este *paraleлизarea execuției instrucțiunilor mașină*, realizată la nivel de instrucțiune mașină. Ea se referă la paralelizarea etapelor execuției unei instrucțiuni, conform Figura 5. Această tehnică de paralelizare se numește *pipelining* (de la cuvântul *pipeline* – conductă).

Prin această tehnică nu crește viteza de execuție a unei instrucțiuni (ea se execută în același număr de cicluri de ceas), ci se mărește numărul de instrucțiuni executate de către procesor per unitate de timp.

Die Ausführungszeit einer Anweisung, gemessen in Prozessortaktzyklen, wird als „*Cycles per Instruction*“ (CPI) bezeichnet. Seit die ersten Prozessoren auf dem Markt waren, wurden die Maschinenbefehle *sequentiell* ausgeführt. Der Prozessor nimmt einen Anweisung entgegen, verbringt bei seiner vollständigen Ausführung eine Anzahl von Taktzyklen und fährt dann mit dem nächsten Anweisung fort und so weiter. Im Allgemeinen bezieht sich der CPI auf die *durchschnittliche* Anzahl von Prozessorzyklen pro Anweisung, die vom Prozessor ausgeführt werden – in Bezug auf alle Anweisungen, die ein Programm bilden, oder in Bezug auf den gesamten Befehlssatz des Prozessors.

Hinweis: Obwohl eine **MOV** Anweisung mit der gleichen Anzahl von N Prozessorzyklen ausgeführt wird, unterscheidet sich die Ausführungszeit (in Sekunden) von Prozessor zu Prozessor. Ein Prozessor, der mit einer Geschwindigkeit von 200 MHz läuft, führt einen **MOV** Anweisung mit 20 Nanosekunden aus, während ein Prozessor, der mit 1 GHz läuft, ihn etwa fünfmal schneller ausführt (4 Nanosekunden).

Da heute eine technologische Grenze hinsichtlich der Taktfrequenz der Prozessoren erreicht ist, soll deren Geschwindigkeit durch andere Methoden gesteigert werden. Am gebräuchlichsten ist die *Parallelisierung der Ausführung der Maschinenbefehle* auf der Ebene der Maschinenbefehle. Es bezieht sich auf die Parallelisierung der Schritte zum Ausführen eines Befehls gemäß Abbildung 5. Diese Parallelisierungstechnik wird *Pipelining* (von der Wort *Pipeline*) genannt.

Diese Technik erhöht nicht die Ausführungs geschwindigkeit einer Anweisung (sie wird in der gleichen Anzahl von Taktzyklen ausgeführt), sondern erhöht die Anzahl der vom Prozessor pro Zeiteinheit ausgeführten Anweisungen.

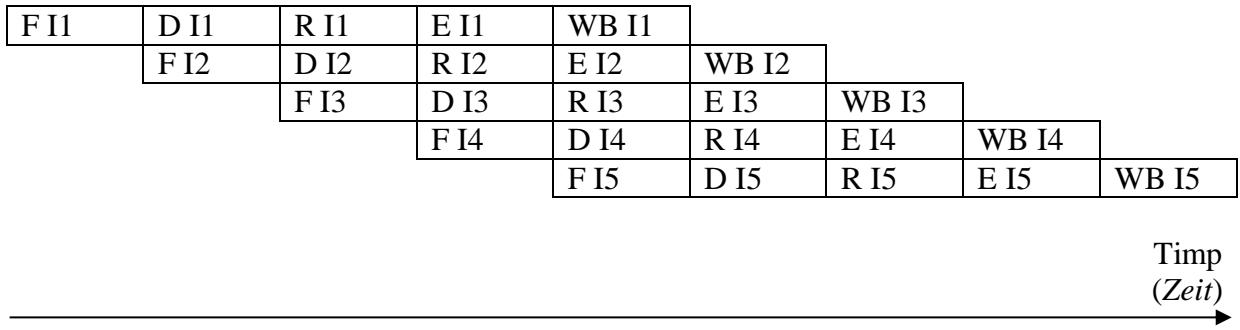


Figura 5. Mecanismul de paralelizare în execuția instrucțiunilor (pipelining) (Der Parallelisierungsmechanismus bei der Ausführung der Anweisungen)

Observații:

1. Ciclurile FETCH, DECODE, READ MEMORY, EXECUTE și WRITE BACK (sau STORE) nu au durate egale, putând varia de la o arhitectură la SC la alta.
2. Numărul de instrucții / ciclu de ceas – *Instructions per Cycle (IPC)* – este un termen care poate crea confuzie dacă ne gândim că CPU poate executa mai mult de o singură acțiune / instrucție în fiecare ciclu de ceas. Tehnica de pipelining permite însă execuția *virtual paralelă* a mai multor instrucții în același timp. Cu cât *pipeline*-ul este mai lung și numărul de cicluri de ceas pentru instrucții mai mic, cu atât crește factorul IPC pentru procesorul respectiv. Pentru exemplul din Figura 6, dacă am considera că fiecare fază F, D, RM, E și WB durează același număr de cicluri de tact (să zicem, 1 ciclu), atunci $IPC = 5 / 9 = 0.55$, iar $CPI = 9 / 5 = 1.8$.
3. Prelungind timpul la infinit și trecând la limită, IPC va tinde la 1.
4. Dacă procesorul ar avea mai multe (în Figura 6, două) unități de execuție și ar prezice corect ce instrucții se pot executa în paralel, ar putea ajunge ca IPC să fie supraunitar. Aceste SC se numesc superscalare. În Figura 6, $IPC =$

Bemerkungen:

1. Die Zyklen FETCH, DECODE, READ MEMORY, EXECUTE und WRITE BACK (oder STORE) haben nicht die gleiche Dauer und können von einer SC-Architektur zur anderen variieren.
2. Anzahl der Anweisungen / Taktzyklus – *Instructions per Cycle (IPC)* – ist ein Begriff, der verwirrend sein kann, wenn wir glauben, dass die CPU in jedem Taktzyklus mehr als eine Aktion / Anweisung ausführen kann. Die Pipelining-Technik ermöglicht jedoch die virtuelle parallele Ausführung mehrerer Befehle gleichzeitig. Je länger die *Pipeline* ist und je kürzer die Anzahl der Taktzyklen für Anweisungen ist, desto höher ist der IPC-Faktor für diesen Prozessor. Wenn wir für das Beispiel in Abbildung 6, berücksichtigen, dass jede Phase F, D, RM, E und WB dieselbe Anzahl von Taktzyklen (z. B. 1 Zyklus) dauert, ist $IPC = 5/9 = 0,55$ und $CPI = 9/5 = 1,8$.
3. Wenn Sie die Zeit bis zum Unendlichen verlängern und das Limit erreichen, tendiert der IPC zu 1.
4. Wenn der Prozessor mehr (in Abbildung 6 zwei) Ausführungseinheiten hat und korrekt voraussagt, welche Anweisungen parallel ausgeführt werden können, ist der IPC möglicherweise zu hoch. Diese CSs werden superskalär genannt. In

$10 / 9 = 1.11$. La ciclul 5 se execută simultan 2 instrucțiuni complete și toate unitățile funcționale ale procesorului sunt solicitate.

Abbildung 6 ist der $IPC = 10/9 = 1.11$. In Zyklus 5 werden 2 vollständige Anweisungen gleichzeitig ausgeführt und alle Funktionseinheiten des Prozessors angefordert.

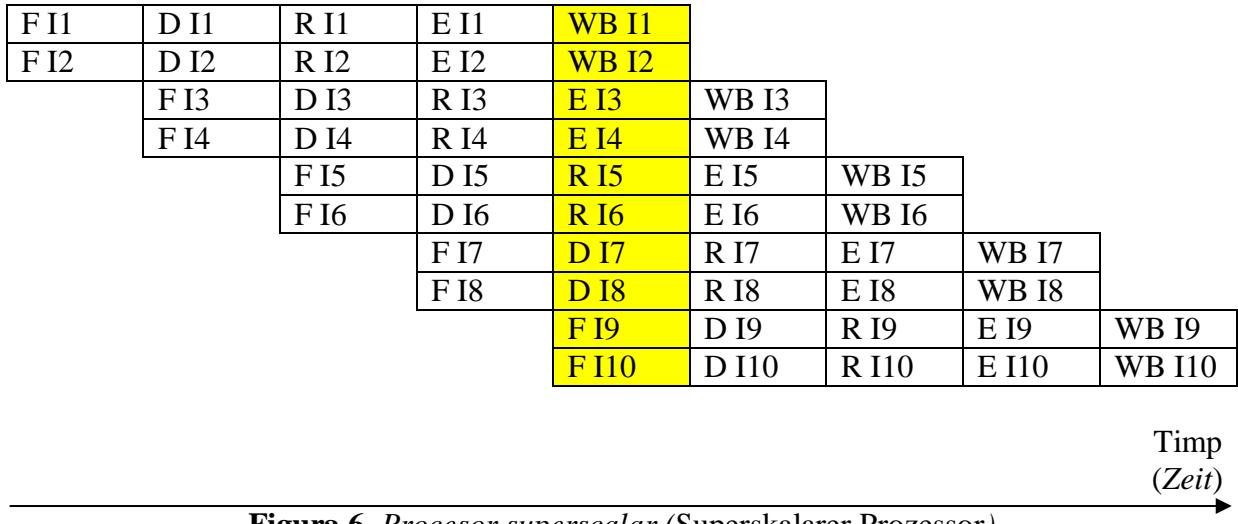


Figura 6. Procesor superscalar (Superskalarer Prozessor)

2.2 Performanțele unui sistem de calcul

2.2.1 Viteza unui sistem de calcul

Pentru măsurarea vitezei calculatoarelor nu se poate folosi numai frecvența de ceas a CPU. Aceasta este o măsură neadecvată, întrucât este foarte importantă și organizarea internă a procesoarelor. S-a introdus apoi noțiunea de MIPS (*Millions of Instructions per Second*) – care nu mai depinde de frecvența tactului. Aceasta măsoară numărul (exprimat în milioane) de instrucțiuni (operând doar pe numere întregi) pe care le poate executa un procesor într-o secundă. MFLOPS (*Millions of Floating-Point Instructions per Second*) reprezintă numărul (exprimat în milioane) de instrucțiuni în virgulă flotantă pe care un procesor le poate executa în unitatea de timp.

În general, un microprocesor este

2.2 Die Leistung eines Computersystems

2.2.1 Die Geschwindigkeit eines Computersystems

Zum Messen der Geschwindigkeit von Computern kann die CPU-Taktfrequenz nicht alleine verwendet werden. Dies ist eine unzureichende Maßnahme, da die interne Organisation der Prozessoren sehr wichtig ist. Dann wurde der Begriff „Millionen von Anweisungen pro Sekunde“ (MIPS (*Millions of Instructions per Second*) eingeführt, der nicht mehr von der Taktfrequenz abhängt. Es misst die Anzahl (ausgedrückt in Millionen) von Anweisungen (die nur für Ganzzahlen ausgeführt werden), die ein Prozessor in einer Sekunde ausführen kann. MFLOPS (*Millions of Floating-Point Instructions per Second*) ist die Anzahl (in Millionen) von Gleitkomma-Anweisungen, die ein Prozessor in der Zeiteinheit ausführen kann.

Im Allgemeinen ist ein Mikroprozessor

caracterizat de:

- viteza de lucru;
- capacitatea maximă de memorie pe care o poate adresa;
- setul de instrucțiuni pe care le poate executa.

Ca și criteriu de performanță se consideră deseori viteza de lucru a CPU, care depinde de:

- frecvența ceasului intern,
- capacitatea de paralelizare (organizarea execuției instrucțiunilor);
- dimensiunea regiștrilor interni;
- lățimea de bandă a magistralei de date;
- tipul microprocesorului sau dimensiunea memoriei cache a CPU (ca factor de influență a vitezei de comunicare cu exteriorul).

2.2.2 Criterii de evaluare a performanțelor

În general sunt utilizate două tipuri de criterii de evaluare a performanțelor:

- timpul de execuție (*Execution Time*) = timpul în care este executată o sarcină, timpul de răspuns;
- rata de execuție (*Throughput*) = numărul de sarcini rezolvate într-un anumit interval de timp.

Rezultă că îmbunătățirea performanței poate fi privită sub două aspecte:

- reducerea timpului de rezolvare a unei sarcini;
- creșterea ratei de execuție.

Acordarea unei „note” întregului sistem de calcul este dificil de realizat, motiv pentru care se face referire la diferite aspecte ale performanțelor componentelor sale. De exemplu:

- CPU:
 - o timpul de execuție (perioadă a tactului mică, execuție paralelă a instrucțiunilor);
 - o rata de execuție (MIPS, MFLOPS, planificarea eficientă a instrucțiunilor);

gekennzeichnet durch:

- Arbeitsgeschwindigkeit;
- die maximale Speicherkapazität, die es adressieren kann;
- den Befehlssatz, den es ausführen kann.

Als Leistungskriterium wird häufig die CPU-Geschwindigkeit herangezogen, die von Folgendem abhängt:

- die Frequenz der internen Uhr,
- Fähigkeit zur Parallelisierung (Organisation der Ausführung von Anweisungen);
- die Größe der internen Register;
- die Bandbreite des Datenbusses;
- den Typ des Mikroprozessors oder die Größe des CPU-Cache (als Faktor, der die Kommunikationsgeschwindigkeit nach außen beeinflusst).

2.2.2 Leistungsbewertungskriterien

Im Allgemeinen werden zwei Arten von Leistungsbewertungskriterien verwendet:

- Ausführungszeit (*Execution Time*) = die Zeit, in der eine Aufgabe ausgeführt wird, die Antwortzeit;
- Durchsatz (*Throughput*) = Anzahl der in einem bestimmten Zeitrahmen gelösten Aufgaben.

Es stellt sich heraus, dass die Leistungsverbesserung kannst auf zwei Arten angesehen sind:

- Verkürzung der Zeit zur Lösung einer Aufgabe;
- Erhöhung der Ausführungsrate.

Das Gewähren einer „Note“ für das gesamte Computersystem ist schwierig zu erreichen, weshalb es sich auf verschiedene Aspekte der Leistung seiner Komponenten bezieht.

Zu Beispiel:

- CPU:
 - o Ausführungszeit (niedrige Taktperiode, parallele Ausführung von Anweisungen);
 - o Ausführungsrate (MIPS, MFLOPS, effektive Anweisungsplanung);

- capacitatea memoriei cache a CPU
- memoria cache: rata de transfer, hit rate vs. miss rate
- memoria internă: capacitatea, rata de transfer
- dispozitivele periferice: viteza de căutare, capacitatea de stocare, viteza de transfer, numărul de pixeli sau poligoane afișate pe secundă
- s.a.m.d.

Alături de performanțele individuale ale componentelor, trebuie avute în vedere și alte aspecte precum: compatibilitatea componentelor, tipurile de date gestionate sau de aplicații executate, sistemul de operare, disponibilitatea software-ului, costurile de proiectare sau costurile de achiziționare sau întreținere.

2.3 „Dimensiunea“ unui microprocesor

Există două perspective:

- perspectiva hardware: lățimea de bandă a magistralei de date (de exemplu: Pentium are o magistrală de date pe 64 biți = 64 linii de date, astfel că la fiecare *memory cycle* procesorul poate accesa 8 octeți din memorie);
- perspectiva software: dimensiunea unui cuvânt de memorie (dimensiunea regiștrilor CPU).

În multe cazuri cele două perspective au coincis ca dimensiune. Diferențe de interpretare apar spre exemplu la procesoarele unde lățimea de bandă a magistralei diferă de lățimea regiștrilor interne.

3. Memoria

Memoria este un dispozitiv de stocare a datelor pentru un anumit interval de timp.
Clasificări:

- CPU-Cache-Kapazität
- Cache-Speicher: Übertragungsrate, Trefferrate vs. Fehlrate
- Interner Speicher: Kapazität, Übertragungsrate
- Peripheriegeräte: Suchgeschwindigkeit, Speicherkapazität, Übertragungsgeschwindigkeit, Anzahl der pro Sekunde angezeigten Pixel oder Polygone
- und so weiter.

Neben den individuellen Leistungen der Komponenten sind weitere Aspekte wie die Kompatibilität der Komponenten, die Art der verwalteten Daten oder ausgeführten Anwendungen, das Betriebssystem, die Verfügbarkeit der Software, die Kosten für das Design oder die Kosten für den Kauf oder die Wartung zu berücksichtigen.

2.3 Die „Größe“ eines Mikroprozessors

Es gibt zwei Perspektiven:

- Hardware-Perspektive: die Bandbreite des Datenbusses (zum Beispiel: Pentium hat einen 64-Bit-Datenbus = 64 Datenleitungen, so dass der Prozessor bei jedem *memory cycle* – Speicherzyklus – auf 8 Byte Speicher zugreifen kann);
- Software-Perspektive: Die Größe eines Speicherworts (die Größe der CPU-Register).

In vielen Fällen stimmten die beiden Perspektiven überein. Interpretationsunterschiede treten beispielsweise bei Prozessoren auf, bei denen sich die Bandbreite des Busses von der Breite der internen Register unterscheidet.

3. Der Speicher

Der Speicher ist ein Gerät, der die Daten für eine bestimmte Zeit speichert.

Klassifizierung:

1. Din punct de vedere al accesării datelor:

- 1.1 *Memorie ROM* (Read-Only Memory), care poate fi accesată numai în citire;
- 1.2 *Memorie RAM, cu acces aleator* (*Random Access Memory*): locațiile pot fi accesate atât în citire, cât și în scriere;
- 1.3 *Memorie cu acces asociativ.* Exemplu: memoria cache.

Observație: La încărcarea sau căutarea în memoria cache a unei informații, pe baza adresei acesteia se determină care este poziția din cache în care trebuie să fie încărcată sau la care ar trebui să se găsească dacă ar exista deja acolo. Strategia de determinare a locației din memoria cache depinde de modul de organizare a acesteia. De exemplu: *maparea directă* – unei informații de la o anumită adresă îi corespunde o locație bine determinată din memoria cache, locație calculată cu ajutorul unei funcții de transformare (poate fi funcție de dispersie de tip modulo); *maparea asociativă pe mulțime* – o anumită informație poate fi încărcată în memoria cache în oricare din pozițiile dintr-o mulțime bine determinată.

1.4 *Memorie cu acces secvențial:* pentru a accesa a n -a înregistrare, trebuie parcuse primele $n-1$ înregistrări => timpul de accesare a datelor este variabil, depinzând de locația accesată. Exemplu: benzi magnetice;

1.5 *Memorie cu acces direct:* spre deosebire de accesul secvențial, poziționarea pe o anumită înregistrare se face în mod direct pe baza unui calcul de adresă. Exemplu: dispozitivele de tip disc, precum hard disk, floppy disk, CD-ROM.

1. Aus Sicht des Datenzugriffs:

- 1.1 *Nur-Lese-Speicher* (*Read-Only Memory ROM*), der nur gelesen werden kann;
- 1.2 *Direktzugriffsspeicher* (*Random Access Memory RAM*): Auf Speicherorte kann sowohl beim Lesen als auch beim Schreiben zugegriffen werden.
- 1.3 *Speicher mit assoziativem Zugriff.* Beispiel: Cache-Speicher.

Hinweis: Beim Hochladen oder Zwischenspeichern von Informationen wird basierend auf ihrer Adresse festgelegt, an welche Cache-Position sie hochgeladen werden sollen oder wo sie sich befinden sollen, wenn sie bereits vorhanden sind. Die Strategie zur Bestimmung des Cache-Speicherorts hängt davon ab, wie sie organisiert ist. Zum Beispiel: *Direkte Zuordnung* – Informationen von einer bestimmten Adresse entsprechen einem gut bestimmten Cache-Speicherort, einem Speicherort, der unter Verwendung einer Transformationsfunktion berechnet wurde (kann eine Funktion der Dispersion vom Modulo-Typ sein); *Assoziatives Mapping auf dem Set* – Eine bestimmte Information kann an jeder Position in einem Set in den Cache Speicher geladen werden.

1.4 *Speicher mit sequenziellem Zugriff:* Um auf den n -ten Datensatz zuzugreifen, müssen die ersten $n-1$ Aufzeichnungen abgeschlossen sein => die Datenzugriffszeit ist variabel, abhängig vom Speicherort, auf den zugegriffen wird. Beispiel: Magnetstreifen;

1.5 *Direktzugriffsspeicher:* Im Gegensatz zum sequentiellen Zugriff erfolgt die Positionierung auf einem bestimmten Datensatz direkt anhand einer Adressberechnung. Beispiel: Datenträgertypen wie Festplatte, Diskette, CD-ROM.

Observație: Modul de accesare a datelor din memoria cu acces aleator este similar cu cel folosit pentru memoria cu acces direct (direct, pe baza unei adrese cunoscute). Diferența constă în timpul de acces: în cazul memoriei cu acces aleator timpul de acces este același, indiferent de adresa datelor accesate; în schimb, memoria cu acces direct folosește un mecanism fizic de poziționare la o anumită adresă, fapt care introduce un timp de întârziere în accesarea datelor => timp de procesare variabil, în funcție de poziția curentă a mecanismului de accesare și de poziția datelor pe disc.

2. Din punct de vedere al volatilității:

2.1 *Memorie volatilă* (de scurtă durată): conținutul său se pierde la îndepărțarea sursei de curent. Aici se încadrează memoria principală a SC (care conține datele și instrucțiunile utilizate curent de CPU);

2.2 *Memorie non-volatile sau remanentă* (de lungă durată): conținutul se păstrează și după deconectarea de la sursă. Exemple: memoria ROM, hard disk, CDROM, memoria Flash.

3. Din punct de vedere al accesului CPU:

3.1. *Memorie internă:* accesată direct de către CPU;

3.2. *Memorie secundară sau dispozitiv de stocare periferic:* memorie externă, cu acces indirect la CPU. Exemple: HD, floppy disk, CDROM.

4. Din punct de vedere al tipurilor de acces permise:

4.1 *Memorie read / write:* permite acces la

Hinweis: Der Datenzugriff aus dem Direktzugriffsspeicher erfolgt auf ähnliche Weise wie der Direktzugriffsspeicher (direkt, basierend auf einer bekannten Adresse). Der Unterschied besteht in der Zugriffszeit: Im Fall eines Direktzugriffsspeichers ist die Zugriffszeit unabhängig von der Adresse der zugegriffenen Daten gleich. Im Gegensatz dazu verwendet der Direktzugriffsspeicher einen physikalischen Positionierungsmechanismus an einer bestimmten Adresse, der eine Zeitverzögerung beim Zugriff auf Daten einführt => variable Verarbeitungszeit, abhängig von der aktuellen Position des Zugriffsmechanismus und der Position der Daten auf der Platte.

2. In Bezug auf die Volatilität:

2.1 *Flüchtiges (Kurzzeit-) Speicher:* Der Inhalt geht verloren, wenn die Stromquelle entfernt wird. Hier befindet sich der Hauptspeicher des CS (der die aktuell von der CPU verwendeten Daten und Anweisungen enthält);

2.2 *Nichtflüchtiger oder verbleibender (Langzeit-) Speicher:* Der Inhalt bleibt nach dem Trennen von der Quelle erhalten. Beispiele: ROM, Festplatte, CD-ROM, Flash-Speicher.

3. Aus Sicht des CPU-Zugriffs:

3.1. *Interner Speicher:* Direktzugriff durch die CPU;

3.2. *Sekundärspeicher oder Peripheriespeicher:* Externer Speicher mit indirektem CPU-Zugriff. Beispiel: HD, Diskette, CD-ROM.

4. Unter dem Gesichtspunkt der zulässigen Zugriffsarten:

4.1 *Lese- / Schreibspeicher:* Ermöglicht den

date în citire sau scriere. Exemple: memoria principală, hard disk, floppy disk;

4.2. *Memorie read-only*: permite doar citirea datelor. Exemple: ROM, CDROM.

Zugriff auf Daten beim Lesen oder Schreiben. Beispiele: Hauptspeicher, Festplatte, Diskette;

4.2. *Nur-Lese-Speicher*: Ermöglicht nur das Lesen von Daten. Beispiel: ROM, CD-ROM.

3.1 Memoria internă

Memoria internă reprezintă toate spațiile de stocare de date accesibile CPU fără utilizarea canalelor de comunicație de intrare / ieșire. Din această categorie fac parte memoria principală, memoria *cache* (dintre CPU și memoria principală), ROM și registrele, toate aceste dispozitive putând fi direct accesate de către CPU.

Memoria principală (*main memory, primary memory*), cunoscută în general sub numele de memorie RAM, conține date care sunt utilizate în mod curent de către procesor – *instructiuni* ale programelor care sunt executate și *date* cu care acestea operează. Aceste informații sunt aduse în memoria principală de pe un suport de stocare extern sau de la un dispozitiv de I/O. Tipurile de memorie care sunt folosite ca memorie principală sunt cele din clasa memoriilor RAM, precum DRAM sau SRAM.

Memoria DRAM (Dynamic RAM) face parte din clasa memoriilor volatile. Datorită modului în care este construită, este necesară reactualizarea conținutului la un anumit interval de timp, de exemplu de ordin μ s (de aici denumirea „dinamică“). Datele nu sunt disponibile în timpul operațiilor de reactualizare. Deși timpul consumat de aceste operații constituie aproximativ 1% din timpul de funcționare, viteza de acces este mai redusă față de alte tipuri de memorie. Conținutul unei asemenea memorii este organizat ca tablou bidimensional de biți. La citirea unui element al tabloului, se citește întreg rândul, care apoi este rescris (*refresh*).

3.1 Interner Speicher

Der interner Speicher ist der gesamte für die CPU zugängliche Datenspeicherplatz ohne Verwendung von Ein- / Ausgabekommunikationskanälen. Diese Kategorie umfasst Hauptspeicher, *Cachespeicher* (zwischen CPU und Hauptspeicher), ROMs und Register, auf die die CPU direkt zugreifen kann.

Der Hauptspeicher (*main memory, primary memory*), allgemein als RAM bezeichnet, enthält Daten, die üblicherweise vom Prozessor verwendet werden - *Anweisungen* für die ausgeführten Programme und *Daten*, mit denen sie arbeiten. Diese Informationen werden von einem externen Speichermedium oder einem E / A-Gerät in den Hauptspeicher übertragen. Die Arten von Speicher, die als Hauptspeicher verwendet werden, sind diejenigen in der RAM-Klasse, wie z. B. DRAM oder SRAM. *DRAM-Speicher (Dynamic RAM)* gehört zur Klasse der flüchtigen Speicher. Aufgrund der Art der Erstellung ist es erforderlich, den Inhalt zu einem bestimmten Zeitpunkt zu aktualisieren, beispielsweise zu μ s (daher der Begriff „dynamisch“). Während der Upgrade-Vorgänge sind die Daten nicht verfügbar. Obwohl die von diesen Vorgängen verbrauchte Zeit etwa 1% der Betriebszeit ausmacht, ist die Zugriffsgeschwindigkeit langsamer als bei anderen Arten von Speichern. Der Inhalt eines solchen Speichers ist als zweidimensionales Array von Bits organisiert. Beim Lesen eines Elements des Gemäldes wird die gesamte Zeile gelesen, die dann neu geschrieben wird (*refresh*). Um

Pentru operația de scriere a unui element, se citește întreg rândul, se modifică elementul, apoi se rescrie întreg rândul înapoi. Elementele unei memorii DRAM sunt mai mici și mai ieftine decât elementele SRAM.

Observație: O alternativă a memoriei DRAM ca organizare este memoria flash, întâlnită în prezent în dispozitive precum carduri de memorie, dispozitive flash USB, camere digitale, telefoane mobile. Acest tip de memorie are un cost per bit mai mic decât al memoriei DRAM, este non-volatile, dar de viteza mai mică la citire / scriere.

Memoria SRAM (Static RAM) este un tip de memorie volatile. După cum indică denumirea, conținutul unei memorii SRAM se păstrează atât timp cât sistemul este conectat la o sursă, spre deosebire de DRAM care necesită reactualizări periodice ale conținutului. Structura SRAM permite un acces mai rapid la locațiile acesteia, în comparație cu DRAM, motiv pentru care este utilizată ca memorie cache a CPU. Memoriile SRAM de viteza și capacitate mai mică sunt folosite atunci când se cere un consum de energie și cost scăzut, de exemplu pentru backup RAM cu sursă de tip baterie. Deoarece este mai puțin densă față de DRAM (conține mai puțini biți pe unitate de suprafață), în general capacitatea unei memorii SRAM este mai mică față de unei memorii DRAM.

Observație: De obicei, raportul de capacitate DRAM/SRAM = 4-8; raportul de cost și timp de acces SRAM/DRAM = 8-16.

Deoarece nu poate fi (ușor) scrisă, memoria ROM este utilizată în general ca spațiu de stocare al *firmware*-ului, care nu necesită actualizări frecvente. *Firmware* reprezintă un ansamblu de resurse software care, prin

ein Element zu schreiben, lesen Sie die gesamte Zeile, ändern Sie das Element und schreiben Sie die gesamte Zeile zurück. DRAM-Speicherelemente sind kleiner und billiger als SRAM-Elemente.

Hinweis: Eine Alternative zum DRAM-Speicher als Organisation ist der Flash-Speicher, der derzeit in Geräten wie Speicherkarten, USB-Flash-Geräten, Digitalkameras und Mobiltelefonen verwendet wird. Diese Art von Speicher hat geringere Kosten pro Bit als DRAM-Speicher, ist nichtflüchtig, jedoch langsamer bei Lese- / Schreibgeschwindigkeit.

SRAM-Speicher (Static RAM) ist eine Art flüchtiger Speicher. Wie der Name andeutet, wird der Inhalt eines SRAM-Speichers so lange gespeichert, wie das System mit einer Quelle verbunden ist, im Gegensatz zu DRAM, das regelmäßige Aktualisierungen des Inhalts erfordert. Die SRAM-Struktur ermöglicht im Vergleich zu DRAM einen schnelleren Zugriff auf ihre Speicherorte, weshalb sie als CPU-Cache verwendet wird. SRAM-Speicher mit niedriger Geschwindigkeit und niedriger Kapazität werden verwendet, wenn ein geringer Stromverbrauch und Kosten erforderlich ist, z. B. für die RAM-Sicherung mit Batteriequelle. Da es weniger dicht ist als DRAM (enthält weniger Bits pro Oberflächeneinheit), ist die Kapazität eines SRAM-Speichers im Allgemeinen kleiner als die eines DRAM-Speichers.

Hinweis: Typischerweise ist das DRAM / SRAM-Kapazitätsverhältnis = 4-8; Kosten- und Zugriffszeitverhältnis SRAM / DRAM = 8-16.

Da es nicht (einfach) beschrieben werden kann, wird das ROM im Allgemeinen als *Firmwares* Speicher verwendet, für den keine häufigen Aktualisierungen erforderlich sind. *Firmware* ist ein Satz von Softwareressourcen, die durch

intermediul microprogramării, preia unele funcții care în mod tradițional revin software-ului (sistemului de operare),

Memoria ROM a multor sisteme de calcul din generațiile trecute (anii '80) conținea încă de la furnizare sistemul de operare, iar o parte din acestea includeau și un interpretor al limbajului de programare BASIC. În prezent, tendința este de a stoca cât mai puține informații în memoriile ROM și o cantitate tot mai mare de date pe dispozitivele de memorare externe. Deși memoria ROM este de capacitate mică, avantajul principal este viteza mare de accesare a datelor. În general este întâlnită ca și componentă CPU, caz în care conține programul de control al acestuia, sau ca suport pentru BIOS. BIOS-ul (*Basic Input/Output System*) este un set de rutine de nivel scăzut care sunt responsabile de inițializarea sistemului, verificarea echipamentelor periferice din sistem și accesul primar la acestea. BIOS-ul poate fi considerat și *firmware*-ul plăcii de bază, rutinele sale fiind printre primele care se execută la pornirea unui calculator.

Memoria ROM este în general cunoscută ca memorie scrisă în faza de producție, al cărui conținut nu poate fi modificat ulterior. Mai există, însă, câteva alte tipuri de memorie ROM al căror conținut poate fi rescris, precum:

- PROM (*Programmable Read-Only Memory*) – poate fi scrisă (programată) o singură dată cu ajutorul unui echipament specializat;
- EPROM (*Erasable Programmable Read-Only Memory*) – permite ștergerea conținutului (prin expunere la ultraviolete) și rescrierea acestuia cu ajutorul unui programator;

Mikroprogrammierung einige Funktionen übernehmen, die traditionell zur Software (Betriebssystem) gehören.

Der ROM-Speicher vieler Rechnersysteme früherer Generationen (1980er Jahre) enthielt bereits das Betriebssystem aus dem Lieferumfang und zum Teil auch einen Interpreter der Programmiersprache BASIC. Gegenwärtig besteht die Tendenz, so wenige Informationen wie möglich in ROMs und eine zunehmende Datenmenge auf externen Speichergeräten zu speichern. Obwohl der ROM-Speicher eine geringe Kapazität aufweist, ist der Hauptvorteil eine hohe Datenzugriffsgeschwindigkeit. Es wird im Allgemeinen als CPU-Komponente gefunden, in welchem Fall es sein Steuerprogramm enthält, oder als BIOS-Unterstützung. Das BIOS (*Basic Input / Output System*) besteht aus einer Reihe von Routinen auf niedriger Ebene, die für die Systeminitialisierung, die Überprüfung der Peripheriegeräte im System und den primären Zugriff darauf verantwortlich sind. Das BIOS kann auch als *Firmware* des Hauptplatinen betrachtet werden, wobei seine Routinen zu den ersten gehören, die beim Starten eines Computers ausgeführt werden.

Der ROM-Speicher wird in der Produktionsphase allgemein als geschriebener Speicher bezeichnet, dessen Inhalt später nicht mehr geändert werden kann. Es gibt jedoch einige andere Arten von ROM-Speichern, deren Inhalt umgeschrieben werden kann, beispielsweise:

- PROM (*Programmable Read Only Memory*) – kann nur einmal mit Hilfe von Spezialgeräten beschrieben (programmiert) werden;
- EPROM (*Erasable Programmable Read Only Memory*) – ermöglicht das Löschen des Inhalts (durch Bestrahlung mit Ultraviolet) und dessen Neuschreiben mit Hilfe eines Programmiergeräts.

- EAROM (*Electrically Alterable Read-Only Memory*) – este folosită în general pentru memorarea permanentă a unor parametri ai sistemului, motiv pentru care este rar modificată; la un moment dat, poate fi alterată o parte a conținutului;
- EEPROM (*Electrically Erasable Read-Only Memory*) – permite ștergerea electrică a întregului conținut sau doar a unui bloc din conținutul memoriei și rescrierea acestuia; exemplu: memoria flash a camerelor digitale, MP3 playerelor și.a.

3.2 Memoria externă / secundără

Memoria secundară reprezintă un dispozitiv de stocare pe termen lung a datelor, care nu sunt curent folosite de către CPU. În general este de capacitate mai mare și are o viteză mai mică de accesare a datelor față de memoria internă și face parte din categoria memoriilor non-volatile.

Câteva dispozitive incluse în această categorie de memorie sunt: hard disk (HDD), floppy disk (FDD), compact disc (CD), DVD, banda magnetică, memoria flash.

Structura unui volum disc

Din punct de vedere fizic, un dispozitiv (volum) disc poate fi alcătuit din unul sau mai multe discuri, plasate concentric pe un ax, în jurul căruia se rotesc cu o viteză constantă. Informația poate fi înregistrată magnetic pe una sau ambele fețe ale unui disc. Pentru accesarea informației, fiecare suprafață de memorare are asociat un cap de citire / scriere. Brațele capetelor de accesare corespunzătoare discurilor sunt situate pe un suport unic al dispozitivului. Mișcarea acestora permite deplasarea capetelor de acces radial pe suprafață

- EAROM (*Electrically Alterable Read-Only Memory*) – wird im Allgemeinen zur dauerhaften Speicherung von Systemparametern verwendet, weshalb es selten geändert wird. Irgendwann kann ein Teil des Inhalts geändert werden.
- EEPROM (*Electrically Erasable Read-Only Memory*) – ermöglicht das elektrische Löschen des gesamten Inhalts oder nur eines Inhaltsblocks des Speichers und dessen Neuschreiben. Beispiel: Flash-Speicher von Digitalkameras, MP3-Playern usw.

3.2 Externer / sekundärer Speicher

Der Sekundärspeicher ist ein Langzeitdatenspeicher, der von der CPU nicht häufig verwendet wird. Es hat im Allgemeinen eine größere Kapazität und eine geringere Datenzugriffsgeschwindigkeit als der interne Speicher und gehört zur Kategorie der nichtflüchtigen Speicher.

Einige Geräte in dieser Kategorie von Speicher sind: Festplatte (HDD), Diskette (FDD), CD (CD), DVD, Magnetband, Flash-Speicher.

Die Struktur eines Datenträgers

Aus physikalischer Sicht kann eine Gerätewand (Volumen-) Platte aus einer oder mehreren Scheiben bestehen, die konzentrisch auf einer Achse angeordnet sind und um die sie sich mit konstanter Geschwindigkeit drehen. Die Informationen können auf einer oder beiden Seiten einer Disc magnetisch aufgezeichnet werden. Um auf die Informationen zuzugreifen, ist jeder Speicheroberfläche ein Lese- / Schreibkopf zugeordnet. Die Zubehörarme der Discs befinden sich auf einer einzigen Halterung des Geräts. Durch ihre Bewegung können

discului, permitând astfel accesarea informației indiferent de localizarea acesteia față de axul central.

Din punct de vedere logic, o suprafață de memorare a discului este divizată în benzi concentrice numite *piste*. Pentru un dispozitiv care deține mai multe suprafete de memorare, numărul de piste de pe aceste suprafete este același, iar pistele de aceeași rază formează un cilindru. O pistă este divizată în porțiuni numite sectoare. Numărul de sectoare este același pentru fiecare pistă a discului și fiecare sector are aceeași dimensiune. Un sector reprezintă în general unitatea de transfer de date între disc și memoria internă.

Parametrii dispozitivului disc sunt următorii: numărul de discuri, numărul de capete de citire / scriere pentru un disc (numărul de fețe active ale unui disc), numărul de piste de pe o față, numărul de sectoare de pe o pistă și numărul de octeți de pe un sector.

O metodă de adresare a informațiilor de pe disc utilizată este CHS (*cylinder-head-sector*). Identificarea sectorului accesat se realizează prin specificarea numărului de ordine al capului de citire / scriere, numărul cilindrului și numărul sectorului din cadrul pistei (unde pistă e determinată de cap și cilindru).

Un criteriu de evaluare a performanței unui dispozitiv disc este timpul de accesare a informației, care poate fi calculat după formula:

$$TimpAcces = TimpCăutare + TimpRotire + TimpTransfer,$$

unde:

- TimpCăutare depinde de numărul de

sich die Enden des radialen Zugriffs auf der Oberfläche der Platte bewegen, wodurch der Zugriff auf Informationen unabhängig von ihrer Position relativ zur Mittelachse ermöglicht wird.

Logischerweise ist eine Plattenspeicherfläche in konzentrische Bänder unterteilt, die als *Spuren* bezeichnet werden. Bei einem Gerät mit mehreren Speicherflächen ist die Anzahl der Spuren auf diesen Flächen gleich und die Spuren mit dem gleichen Radius bilden einen Zylinder. Eine Spur ist in Abschnitte unterteilt, die als Sektoren bezeichnet werden. Die Anzahl der Sektoren ist für jede Platten Spur gleich und jeder Sektor hat die gleiche Größe. Ein Sektor ist im Allgemeinen die Einheit für die Datenübertragung zwischen Festplatte und internem Speicher.

Die Plattengeräteparameter lauten wie folgt: Anzahl der Platten, Anzahl der Lese- / Schreibenden für eine Platte (Anzahl der aktiven Seiten einer Platte), Anzahl der Spuren auf einer Seite, Anzahl der Sektoren auf einer Spur und Anzahl der Bytes von auf einem Sektor.

Ein Verfahren zum Adressieren der verwendeten Platteninformation ist CHS (*cylinder-head-sector*, Zylinderkopfsektor). Die Identifizierung des Sektors, auf den zugegriffen wird, erfolgt durch Angabe der Ordnungsnummer des Lese- / Schreibkopfs, der Nummer des Zylinders und der Nummer des Sektors innerhalb der Spur (wobei die Spur durch den Kopf und den Zylinder bestimmt wird).

Ein Kriterium für die Bewertung der Leistung eines Plattengeräts ist der Zeit des Zugriffs auf die Informationen, der sich nach folgender Formel berechnen lässt:

$$\text{Zugriffszeit} = \text{Suchzeit} + \text{Rotationszeit} + \text{Übertragungszeit},$$

wo:

- piste peste care trebuie să deplaseze capul de accesare și viteza de căutare a discului. Aici putem adăuga și TimpController, care reprezintă întârzierea produsă de controlerul dispozitivului (acesta având rol de interfață de comunicare între CPU și disc);
- TimpRotire este în funcție de viteza de rotație a discului și de distanța dintre sectorul care trebuie accesat și capul de accesare;
 - TimpTransfer depinde de rata de transfer a datelor (*bandwidth*) caracteristică dispozitivului.
- Die Suchzeit hängt von der Anzahl der Titel ab, über die sich der Zugriffskopf und die Suchgeschwindigkeit der Festplatte bewegen müssen. Hier können wir auch ControllerZeit hinzufügen, der die Verzögerung darstellt, die durch den Gerätecontroller verursacht wird (dieser hat die Rolle der Kommunikationsschnittstelle zwischen der CPU und der Festplatte).
- Rotationszeit basiert auf der Rotationsgeschwindigkeit der Platte und der Entfernung zwischen dem Sektor, auf den zugegriffen werden soll, und dem Zugriffskopf.
- Übertragungszeit ist abhängig von der Datenübertragungsrate (*bandwidth*, Bandbreite) des Geräts.

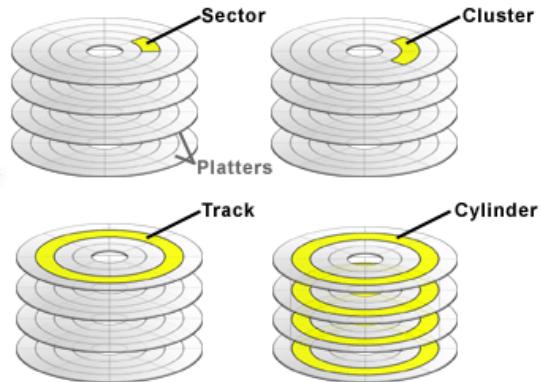
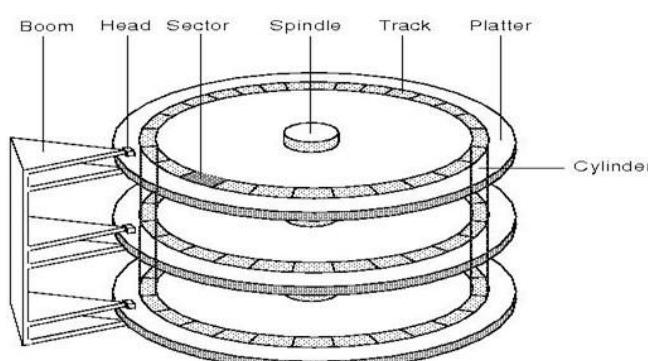


Figura 7. Structura fizică a unui hard disc (Physikalische Struktur einer Festplatte)

Cele mai utilizate volume disc sunt:

- hard disk-urile – dispozitive de stocare a datelor pe suport magnetic, alcătuite din mai multe discuri; capacitatea de memorare a acestora este de ordinul sutelor de gigabytes;
- benzile magnetice – permit acces secvențial la date, asigură o capacitate mare de stocare, sunt ieftine și sunt folosite în general pentru arhivare și memorarea copiilor de siguranță (suport de backup);
- discurile optice:

Die am häufigsten verwendeten Datenträger sind:

- Festplatten - Geräte zum Speichern von Daten auf magnetischen Medien, die aus mehreren Festplatten bestehen; Ihre Speicherkapazität liegt in der Größenordnung von Hunderten von Gigabyte.
- Magnetstreifen – ermöglichen den sequentiellen Zugriff auf Daten, bieten eine hohe Speicherkapazität, sind kostengünstig und werden in der Regel zum Archivieren und Speichern von Backups (Backup-Medien) verwendet.
- optische Disks:

- CD (Compact Disc) – dispozitive de capacitate de memorare relativ mare (650-700 MB), de tip Read-Only sau Read / Write, pentru care producția și duplicarea nu sunt costisitoare;
- DVD (*Digital Versatile Disc*) – dispozitive disc similare CD-urilor, însă prezintă un mod de codificare a datelor diferit și o densitate a acestora mai mare; capacitatea de stocare a acestora este în prezent de 4.7 GB până la 17.1 GB; pot fi de tip Read Only (DVD-ROM) sau Read / Write (DVD-RW);
- Alte tipuri de discuri optice: Blu-Ray Disc, High Density Digital Versatile Disc (HD DVD) etc.
- CD (Compact Disc) – Geräte mit relativ großer Speicherkapazität (650-700 MB) vom Typ Read-Only oder Read / Write, für die Produktion und Vervielfältigung nicht teuer sind;
- DVD (*Digital Versatile Disc*) – Geräte, die CDs ähneln, jedoch einen anderen Datencodierungsmodus und eine höhere Dichte aufweisen. Ihre Speicherkapazität beträgt derzeit 4,7 GB bis 17,1 GB. Sie können Read Only (DVD-ROM) oder Read / Write (DVD-RW) sein.
- Andere Arten von optischen Discs: Blu-Ray-Discs, High Density Digital Versatile Discs (HD-DVDs) usw.

3.3 Ierarhia memoriei

Diferența de performanță dintre componentele unui SC poate fi semnificativă. Pentru a gestiona cât mai eficient accesul la date, un sistem de calcul trebuie să aibă un sistem complex de memorie, în care combină memorii de capacitate mică, dar rapide, și memorii de capacitate mare, însă de viteză redusă. Drept rezultat, un asemenea sistem se comportă în general ca o memorie rapidă, de capacitate mare. Nivelurile ierarhice ale unui asemenea sistem pot fi reprezentate astfel (Figura 8):

3.3 Die Speicherhierarchie

Der Leistungsunterschied zwischen den Komponenten eines CS kann erheblich sein. Um den Datenzugriff so effizient wie möglich zu verwalten, verfügt ein Computersystem über ein komplexes Speichersystem, in dem kleiner, aber schneller Speicher und größer, aber langsamer Speicher kombiniert werden. Infolgedessen verhält sich ein solches System im Allgemeinen wie ein schneller Speicher mit hoher Kapazität. Die hierarchischen Ebenen eines solchen Systems können wie folgt darstellen (Abbildung 8):

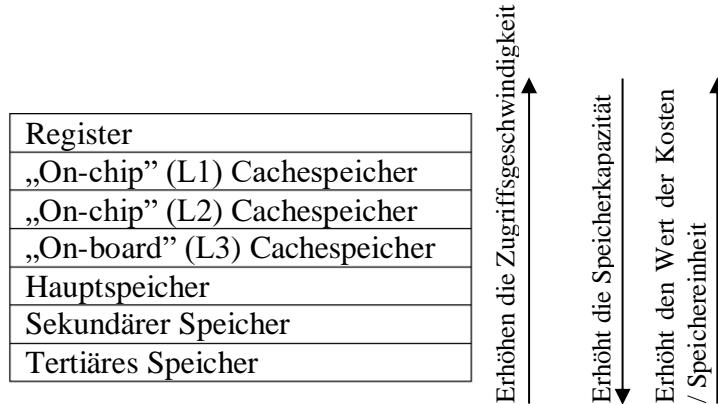


Figura 8. Ierarhia memoriei (Die Speicherhierarchie)

Se observă că nivelurile de memorie de capacitate mai mică se găsesc mai aproape de procesor decât memoriile de capacitate mare, dar de viteză de acces mai mică.

În general, un nivel al ierarhiei reprezintă o submulțime de informații a unui nivel inferior: datele care se găsesc pe primul nivel sunt aduse din următorul nivel de memorie, mai îndepărtat de CPU. Pentru a gestiona un asemenea trafic al datelor între diferite niveluri este nevoie de funcții de transformare a adreselor de pe nivelul inferior către cel imediat superior. Întotdeauna datele sunt copiate numai între două niveluri adiacente.

Eficiența unui asemenea sistem este asigurată de *principiul localizării*:

- *localizare temporală*: după accesarea unei date sunt mari şanse ca ea să fie accesată din nou în scurt timp => ar trebui să se mai rețină data respectivă pentru o perioadă de timp (de exemplu: instrucțiunile dintr-o structură repetitivă sau ale unei subrutine);
- *localizare spațială*: dacă se accesează o locație, sunt mari şanse să urmeze accesarea unor locații din vecinătatea

Es ist zu erkennen, dass Speicherebenen mit geringerer Kapazität näher am Prozessor liegen als Speicher mit hoher Kapazität, jedoch mit geringerer Zugriffsgeschwindigkeit.

Im Allgemeinen repräsentiert eine Hierarchieebene eine Teilmenge von Informationen einer niedrigeren Ebene: Die auf der ersten Ebene gefundenen Daten werden von der nächsten Speicherebene weiter von der CPU entfernt gebracht. Um diesen Datenverkehr zwischen verschiedenen Ebenen zu verwalten, ist es erforderlich, die Adressen von der niedrigeren auf die unmittelbar höhere Ebene umzuwandeln. Die Daten werden immer nur zwischen zwei benachbarten Ebenen kopiert.

Die Effizienz eines solchen Systems wird durch das *Prinzip der Lokalisierung* sichergestellt:

- *Temporärer Speicherort*: Nach dem Zugriff auf Daten besteht die Möglichkeit, dass in kurzer Zeit erneut auf die Daten zugegriffen wird. => Das Datum sollte für einen bestimmten Zeitraum beibehalten werden (z. B. Anweisungen in einer sich wiederholenden Struktur oder einem Unterprogramm);
- *räumlicher Lage*: Wenn auf einen Ort zugegriffen wird, ist es wahrscheinlich, dass die Zugriffe auf Orte in der Nähe

primeia => ar trebui ca la accesarea datei curente să se aducă un întreg bloc de informație care să conțină atât informația necesară în momentul curent, cât și informația conținută la adrese începătore (de exemplu: variabile locale unei subrute sau elementele unui sir).

În urma acestor observații statistice s-au dezvoltat aşa-numitele memorii de tip cache.

3.3.1. Memoria cache

O memorie de tip cache este o colecție de date care reprezintă duplicarea valorilor originale stocate într-un alt tip de dispozitiv de memorare, a căror accesare pentru citire / procesare este mai costisitoare (ca timp) decât accesarea lor din cache. Memoria cache are capacitate mai mică, însă oferă un timp de acces la date cu mult mai rapid față de timpul asigurat de componenta asociată.

Odată ce datele sunt aduse în memoria cache, ele vor fi accesate de aici, fără a fi nevoie de repetarea operației de copiere, scăzând astfel semnificativ timpul mediu de accesare.

Memoria cache exploatează localitatea spațială și temporală. O putem întâlni ca interfață între diferite niveluri ale ierarhiei memoriei sau în asociere cu diferite dispozitive periferice sau chiar componente soft. Astfel, în prezent noțiunea de memorie cache reprezintă o tehnică de optimizare a accesului la date, indiferent de tipul clientului cache pe care îl deservește (memorie, dispozitiv periferic sau componentă software – sistem de operare sau aplicație utilizator). În general, însă, se face referință la memoria cache ca fiind interfață dintre CPU și memoria principală.

des ersten folgen => es sollte auf das aktuelle Datum gebracht werden, um einen vollständigen Informationsblock zu bringen, der sowohl die zum aktuellen Zeitpunkt als auch die benötigten Informationen enthält Informationen, die in benachbarten Adressen enthalten sind (z. B. lokale Variablen eines Unterprogramms oder Elemente eines Strings).

Infolge dieser statistischen Beobachtungen entwickelten sich die sogenannten Cache-Speicher.

3.3.1 Der Cache-Speicher

Ein Cache-Speicher ist eine Sammlung von Daten, die ein Duplikat der Originalwerte darstellt, die in einem anderen Typ von Speichergerät gespeichert sind, dessen Lese-/Verarbeitungszugriff zeitaufwendiger ist als der Cache-Speicher Zugriff. Der Cache-Speicher hat eine geringere Kapazität, bietet jedoch eine schnellere Zugriffszeit auf Daten als die von der zugeordneten Komponente bereitgestellte Zeit.

Sobald die Daten in dem Cache-Speicher sind, wird von hier aus auf sie zugegriffen, ohne dass der Kopiervorgang wiederholt werden muss, wodurch die durchschnittliche Zugriffszeit erheblich verkürzt wird.

Der Cache-Speicher nutzt die räumliche und zeitliche Lokalität aus. Wir können es als Schnittstelle zwischen verschiedenen Ebenen der Speicherhierarchie oder in Kombination mit verschiedenen Peripheriegeräten oder sogar Softwarekomponenten finden. Daher ist der Cache-Speicher gegenwärtig eine Technik zur Optimierung des Zugriffs auf Daten, unabhängig vom Typ des Cache-Clients, den er bedient (Speicher, Peripheriegerät oder Softwarekomponente - Betriebssystem oder Benutzeranwendung). Im Allgemeinen wird der Cache-Speicher jedoch als Schnittstelle zwischen der CPU

Exemplu: interfața dintre CPU și memoria internă de tip DRAM. Aceasta poate fi alcătuită din unul, două sau trei niveluri de memorie cache. O mare parte a sistemelor de calcul din prezent folosesc memorie cache pe două niveluri (L1 și L2). Primul nivel de cache este integrat pe chip-ul CPU (cache „on-chip”) și asigură o viteză de acces similară CPU. Al doilea nivel asigură interfața dintre primul nivel și memoria internă și în general este memorie de tip SRAM. Este plasată de obicei tot pe chip-ul CPU, însă viteză de acces este mai redusă decât cea asigurată L1 cache, iar capacitatea de stocare este mai mare. În cazul în care există și un al treilea nivel de memorie cache (L3), aceasta este amplasată pe placă de bază (cache „on-board”).

Secvența de căutare este:

- CPU solicită o instrucțiune sau un operand; să notăm cu I informația solicitată.
- Se caută în primul nivel cache, cel mai apropiat de CPU.
- Dacă I este găsită \Rightarrow cache hit (succes); se oprește căutarea pe acest nivel.
- Dacă I nu este găsită \Rightarrow cache miss (eșec); se continuă căutarea pe nivelul cache secundar
- În cazul în care pe toate nivelurile cache s-a raportat eșec, se caută I în memoria principală.

Criterii de măsură a performanței unei memorii cache: $hit\ rate = \frac{nr_hit}{nr_referințe}$ la memorie (procentul de accesări cu succes din totalul de accesări); $miss\ rate = \frac{nr_miss}{nr_referințe}$ la memorie (procentul de tentative de acces eşuate din totalul de cereri), unde $miss\ rate = 1 - hit\ rate$.

Principalele tipuri de memorie cache:

und dem Hauptspeicher bezeichnet.

Beispiel: Die Schnittstelle zwischen der CPU und dem internen DRAM-Speicher. Dies kann aus einer, zwei oder drei Ebenen des Cache-Speichers bestehen. Die meisten heutigen Computersysteme verwenden einen zweistufigen Cache (L1 und L2). Die erste Cache-Ebene ist auf dem CPU-Chip integriert („On-Chip“ -Cache) und bietet eine ähnliche CPU-Zugriffsgeschwindigkeit. Die zweite Ebene bildet die Schnittstelle zwischen der ersten Ebene und dem internen Speicher und ist im Allgemeinen ein Speicher vom Typ SRAM. Es wird normalerweise auch auf dem CPU-Chip platziert, aber die Zugriffsgeschwindigkeit ist langsamer als der gesicherte L1-Cache und die Speicherkapazität ist höher. Wenn es eine dritte Cache-Ebene (L3) gibt, befindet sie sich auf der Hauptplatine („On-Board“ -Cache).

Die Suchreihenfolge ist:

- Die CPU fordert eine Anweisung oder einen Operanden an. Wir notieren mit I die angeforderten Informationen.
- Wir suchen nach der ersten Cache-Ebene, die der CPU am nächsten liegt.
- Wenn I gefunden werde \Rightarrow Cache-Treffer (cache hit – Erfolg); Die Suche auf dieser Ebene stoppt.
- Wenn I nicht gefunden werde \Rightarrow Cache fehlgeschlagen (cache miss); weiterhin auf der sekundären Cache-Ebene suchen,
- Wenn ein Fehler auf allen Cache-Ebenen gemeldet wurde, durchsuche ich den Hauptspeicher.

Kriterien zur Messung der Leistung eines Caches: Trefferrate ($hit\ rate$) = $\frac{no_hit}{no_reference\ to\ memory}$ (der Prozentsatz erfolgreicher Treffer von den gesamten Treffern); $miss\ rate = \frac{no_miss}{no_reference\ to\ memory}$ (Prozentsatz der fehlgeschlagenen Zugriffsversuche von der Gesamtzahl der Anforderungen), wobei $miss\ rate = 1 - hit\ rate$.

Die Hauptarten des Cache-Speichers:

- cache al memoriei principale, ca interfață între aceasta și CPU: poate fi pe unul, două sau trei niveluri; acest cache este gestionat de către hardware;
- cache între memoria principală și memoria secundară (disc) este memoria virtuală; transferul datelor de pe disc în memoria principală (gestiunea memoriei virtuale) este gestionat de sistemul de operare;
- cache *Translation Lookaside Buffer* (TLB) al tabelei de pagini folosită pentru realizarea corespondenței de adrese între memoria principală și memoria virtuală;
- memorii cache gestionate de componente soft: cache DNS (pentru corespondențe dintre nume de domenii și adrese IP); cache al unui web browser (pentru ultimele pagini accesate); cache al unui SGBD (de exemplu: Oracle, SQL-Server, pentru ultimele date citite sau ultimele planuri de execuție dezvoltate).

În general, performanța unei memorii cache depinde de parametrii de organizare și funcționare ai acestora, precum: algoritmul de plasare a blocurilor aduse în cache (*block placement strategy*), modalitatea de identificare a unui bloc din cache (*block identification policy*), politica de selectare a unui bloc care să fie înlocuit în cazul în care nu mai este spațiu liber în cache (*block replacement policy*), scrierea datelor modificate (*cache write policy* – imediat sau la dealocarea blocului).

- Hauptspeicher-Cache als Schnittstelle zwischen ihm und der CPU: Er kann sich auf einer, zwei oder drei Ebenen befinden. Dieser Cache wird von der Hardware verwaltet.
- Cache zwischen Hauptspeicher und Sekundärspeicher (Festplatte) ist virtueller Speicher; Die Datenübertragung von der Festplatte zum Hauptspeicher (Verwaltung des virtuellen Speichers) wird vom Betriebssystem verwaltet.
- Cache *Translation Lookaside Buffer* (TLB) der Seitentabelle, die für den Adressabgleich zwischen Hauptspeicher und virtuellem Speicher verwendet wird;
- Software-verwaltete Caches: DNS-Cache (für Übereinstimmungen mit Domänennamen und IP-Adressen); Cache eines Webbrowsers (für die zuletzt aufgerufenen Seiten); Cache eines DBMS (zum Beispiel: Oracle, SQL-Server, für die zuletzt gelesenen Daten oder die neuesten entwickelten Ausführungspläne).

Im Allgemeinen hängt die Leistung eines Cache-Speichers von seiner Organisation und den Funktionsparametern ab, z. B.: der Blockplatzierungsstrategie (*block placement strategy*), der Methode zum Identifizieren einer Block (Blockidentifizierungsrichtlinie *block identification policy*) von der Cache und der Auswahlrichtlinie eines Blocks, der ersetzt werden soll, wenn im Cache kein freier Speicherplatz mehr vorhanden ist (*block replacement policy* Blockersetzungsrtschlinie), Schreiben der geänderten Daten (*cache write policy* – sofort oder wenn der Block blockiert ist).

4. Dispozitivele periferice

Dispozitivele periferice asigură interfață dintre utilizator și sistemul de calcul sau dintre sistemul de calcul și alte sisteme fizice. Sunt caracterizate de tipul de

4. Die Peripheriegeräte

Die Peripheriegeräte stellen die Schnittstelle zwischen dem Benutzer und dem Computersystem oder zwischen dem Computersystem und anderen physischen

comportament (intrare, ieșire, stocare), partener (utilizator uman sau sistem fizic) și performanță.

Performanța unui dispozitiv de intrare / ieșire (Input / Output – I/O) depinde de tipul său și de alte componente ale sistemului (CPU, memorii, magistrale, controler și software I/O, sistem de operare) și este reprezentată de rata de transfer a datelor (*I/O bandwidth* = cantitatea de date transmisă și recepționată într-un interval de timp) și timpul de răspuns al dispozitivului periferic (*latency*).

Categorii de dispozitive periferice:

- dispozitive de intrare: tastatură, mouse, scanner;
- dispozitive de ieșire: imprimantă, monitor;
- dispozitive de intrare sau ieșire: modem, placă de rețea;
- dispozitive de stocare: disc (hard disk, floppy disk), bandă magnetică.

4.1 Magistrale – structuri de interconectare

O magistrală este un subsistem prin care se transportă informație (date, instrucțiuni, semnale de control) sau energie între diferite componente ale unui SC sau între diferite SC. Spre deosebire de conexiunile punct la punct, o singură magistrală poate realiza o conexiune între două sau mai multe componente.

În sistemele de calcul moderne magistrala poate fi de tip paralel sau serial. Prin magistralele seriale se transportă informația ca sir de biți (bit după bit). Magistralele paralele transportă simultan informație prin mai multe fire, mărindu-se astfel rata de transfer.

Acest lucru nu înseamnă că pe o magistrală paralelă viteza de transfer a informației va

fi mai mare decât pe o magistrală serială. Viteza de transfer depinde de caracteristicile sistemelor care sunt conectate la magistrală, de modul de funcționare (Eingabe, Ausgabe, Speicherung), de partener (menschlicher Benutzer oder physisches System) și de performanță.

Die Leistung eines Eingabe- / Ausgabegeräts (Eingabe / Ausgabe - E/A) hängt von seinem Typ und anderen Systemkomponenten (CPU, Speicher, Bus, Controller und Software-E/A, Betriebssystem) ab und wird durch die Rate von dargestellten Datenübertragung (E/A-Bandbreite *I/O bandwidth* = über einen bestimmten Zeitraum übertragene und empfangene Datenmenge) und Antwortzeit des Peripheriegeräts (*Latenz*).

Kategorien von Peripheriegeräten:

- Eingabegeräte: Tastatur, Maus, Scanner;
- Ausgabegeräte: Drucker, Monitor;
- Eingabe- oder Ausgabegeräte: Modem, Netzwerkkarte;
- Speichergeräte: Diskette (Festplatte, Diskette), Magnetband.

4.1 Busse - Verbindungsstrukturen

Ein Bus ist ein Subsystem, über das Informationen (Daten, Anweisungen, Steuersignale) oder Energie zwischen verschiedenen Komponenten eines CS oder zwischen verschiedenen CSs transportiert werden. Im Gegensatz zu Punkt-zu-Punkt-Verbindungen kann ein einzelner Bus eine Verbindung zwischen zwei oder mehr Komponenten herstellen.

In modernen Computersystemen kann der Bus vom parallelen oder seriellen Typ sein. Die seriellen Busse übertragen die Informationen als Bitfolge (Bit für Bit). Parallele Busse übertragen Informationen gleichzeitig über mehrere Leitungen, wodurch die Übertragungsrate erhöht wird. Dies bedeutet nicht, dass auf einem parallelen Bus die Geschwindigkeit der

fi neapărat mai mare decât pe una serială. Dimpotrivă, datorită costurilor mari implicate de transmisia paralelă a datelor, în ultimul timp se remarcă renunțarea la magistralele paralele și concentrarea pe magistrale seriale care să lucreze la frecvențe de transfer mari (de exemplu: magistrala serială S-ATA are o frecvență de transfer mai mare decât magistrala paralelă IDE/ATA; similar pentru interfața serială USB în raport cu oricare interfață paralelă aflată în uz).

Un sistem de calcul include magistrale interne (locale) care fac legătura între componente interne ale sistemului (de exemplu: între CPU și memoria internă) și magistrale externe, pentru realizarea conexiunilor către echipamente periferice sau către alte mașini.

Sub-sistemul de magistrale al unui SC poartă numele de magistrală sistem (*system bus*). Raportat la tipul informației transportate, pot fi identificate trei tipuri de magistrale într-un SC:

- *magistrale de date (data bus)* – transportă informație de tip dată sau instrucțiune. Performanța depinde în primul rând de lățimea de bandă (de exemplu: magistrale de 8, 16, 32, 64 biți);
- *magistrale de adrese (address bus)* – informația comunicată este adresa locației de memorie pe care componenta solicitantă dorește să o acceseze (în citire sau scriere). Lățimea de bandă a magistralei determină capacitatea maximă de memorie adresabilă din sistem (de exemplu: sistemele de calcul cu registre pe 8 biți cu magistrale pe 16 biți, de unde rezultă $2^{16} = 64\text{K}$ locații de memorie adresabile, iar sistemele PC mai noi au magistrale pe 32 biți - $2^{32} = 4\text{GB}$ locații);
- *magistrale de control (control bus)* –

Informationsübertragung notwendigerweise höher als auf einem seriellen Bus ist. Im Gegenteil, aufgrund der hohen Kosten, die mit der parallelen Übertragung von Daten verbunden sind, wird in letzter Zeit der Verzicht auf parallele Busse und der Fokus auf serielle Busse, die mit hohen Übertragungsfrequenzen arbeiten, bemerkt (zum Beispiel: der serielle S-ATA-Bus hat eine Frequenz von Übertragung größer als der parallele IDE / ATA-Bus; ähnlich wie die serielle USB-Schnittstelle im Vergleich zu jeder verwendeten parallelen Schnittstelle).

Ein Computersystem umfasst interne (lokale) Busse, die die internen Komponenten des Systems (z. B. zwischen CPU und internem Speicher) und externe Busse verbinden, um Verbindungen zu Peripheriegeräten oder anderen Maschinen herzustellen.

Das Subsystem von Bussen eines CS wird als Systembus (*system bus*) bezeichnet. Abhängig von der Art der übermittelten Informationen können in einem CS drei Arten von Bussen identifiziert werden:

- *Datenbusse (data bus)* – Übertragen von Daten- oder Anweisungsinformationen. Die Leistung hängt in erster Linie von der Bandbreite ab (z. B. 8, 16, 32, 64-Bit-Busse).
- *Adressbusse (address bus)* – Die übermittelte Information ist die Adresse des Speicherorts, auf den die anfordernde Komponente zugreifen möchte (beim Lesen oder Schreiben). Die Busbandbreite bestimmt die maximal adressierbare Speicherkapazität im System (zum Beispiel: 8-Bit-Registercomputersysteme mit 16-Bit-Bussen, die $2^{16} = 64\text{ KB}$ adressierbare Speicherorte ergeben, und neuere PC-Systeme mit eingeschalteten Bussen 32 Bit - $2^{32} = 4\text{ GB}$ Speicherorte);
- *Steuerbusse (control bus)* – Übertragung von Steuer- und

transmit informații de control și semnalizare (de exemplu: semnale de citire / scriere a memoriei, cerere de utilizare a magistralei de date acceptarea cererii de utilizare a magistralei, semnale de ceas, *reset*).

În funcție de tipul componentelor interconectate, magistralele interne pot fi:

- magistrale CPU-memorie: asigură o comunicare rapidă directă între procesor și memorie și sunt de lungime redusă;
- magistrale de I/O: au în general arhitectura standardizată și asigură o viteză ridicată în comunicarea informației dintre CPU, memorie și un controler de I/O.

Cele mai cunoscute tipuri de magistrale de I/O: ISA (*Industry Standard Architecture*), PCI (*Peripheral Component Interconnect*), PCI Express și AGP (*Accelerated Graphics Port*). Aceste magistralele de I/O permit comunicarea cu dispozitivul periferic prin intermediul unui controler.

Fizic, un controler este o placă de extensie atașabilă sistemului de calcul. De obicei, fiecare controler prezintă două interfețe:

- o interfață de comunicare cu procesorul prin intermediul magistralei de I/O. În funcție de tipul de magistrală folosit, această interfață poate fi de exemplu ISA, PCI sau AGP;
- o interfață de comunicare cu echipamentul periferic propriu zis. Această interfață diferă de la un controler la altul în funcție de echipamentul periferic care îl este atașat.

Signalisierungsinformationen (z. B. Lese- / Schreibspeichersignale, Anforderung zur Verwendung des Datenbusses, Annahme der Anforderung zur Verwendung des Busses, Taktsignale, *reset*).

Abhängig von der Art der miteinander verbundenen Komponenten können die internen Busse sein:

- CPU-Speicherbusse: sorgen für eine schnelle direkte Kommunikation zwischen Prozessor und Speicher und sind von geringer Länge;
- E / A-Busse: Sie haben im Allgemeinen eine standardisierte Architektur und bieten eine hohe Geschwindigkeit bei der Kommunikation von Informationen zwischen CPU, Speicher und einem E / A-Controller.

Die beliebtesten Arten von E / A-Bussen: ISA (*Industry Standard Architecture*), PCI (*Peripheral Component Interconnect*), PCI Express und AGP (*Accelerated Graphics Port*). Diese E / A-Busse ermöglichen die Kommunikation mit dem Peripheriegerät über eine Steuerung.

Ein Controller ist eine Erweiterungskarte, die an das Computersystem angeschlossen werden kann. Normalerweise hat jeder Controller zwei Schnittstellen:

- eine Kommunikationsschnittstelle zum Prozessor über den E / A-Bus. Abhängig von der Art des verwendeten Busses kann diese Schnittstelle zum Beispiel ISA, PCI oder AGP sein;
- eine Kommunikationsschnittstelle zum Peripheriegerät selbst. Diese Schnittstelle unterscheidet sich von Controller zu Controller in Abhängigkeit von den angeschlossenen Peripheriegeräten.

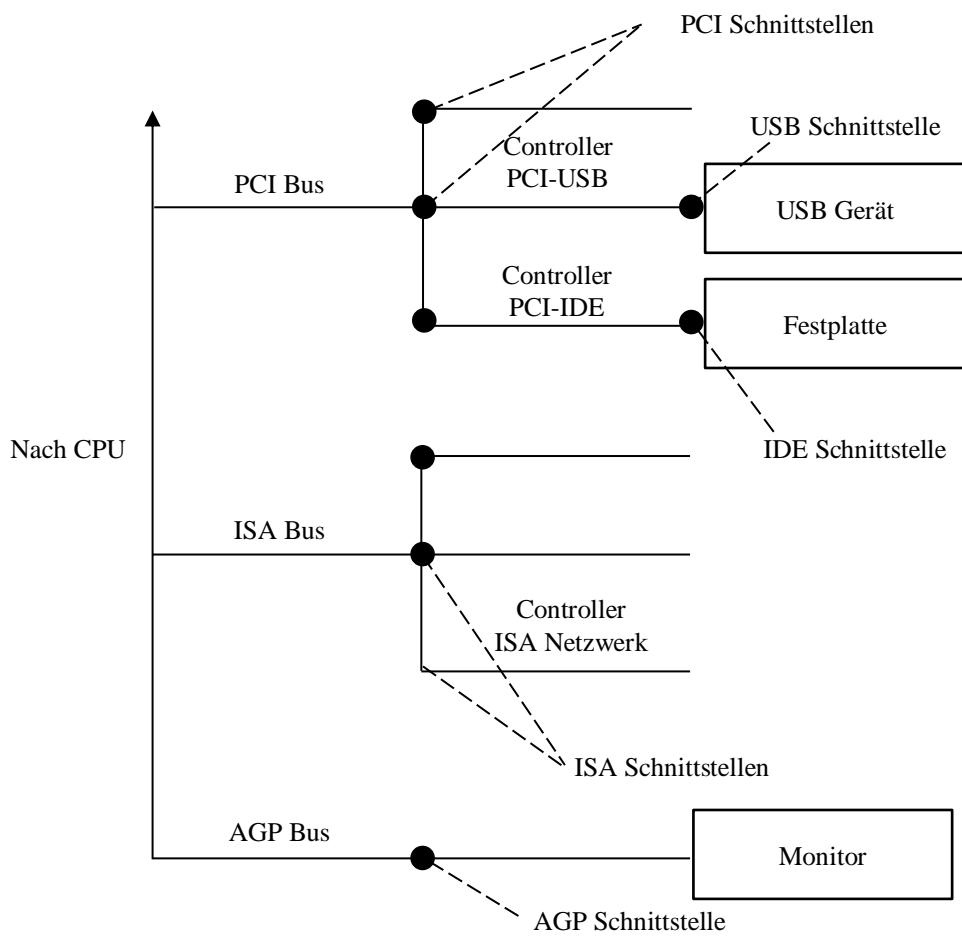


Figura 9. Magistrale de I/O (E/A Busse)

4.2. Magistrale I/O interne și interfețele asociate

4.2.1 Magistrala ISA

A fost introdusă de IBM la începutul anilor '80, rezistând până la sfârșitul anilor '90. ISA a fost dezvoltată mai întâi pe 8 biți, iar ulterior pe 16 biți, operând la 8 MHz. Aceste valori au fost potrivite pentru dimensiunea magistralei sistem și frecvența procesorului 286, permitând viteze de până la 16 Megaocăți / secundă. În timp, viteză oferită de o magistrală ISA a devenit insuficientă. Deși, teoretic, pe o magistrală ISA se pot obține viteze de până la 16 Megaocăți/secundă, vitezele reale sunt

4.2. Interne E / A-Busse und die zugehörigen Schnittstellen

4.2.1 ISA-Bus

Es wurde von IBM in den frühen 80er Jahren eingeführt und widerstand bis in die späten 90er Jahre. Der ISA wurde zuerst mit 8 Bit und dann mit 16 Bit entwickelt und mit 8 MHz betrieben. Diese Werte waren für die Größe des Systembusses und die Prozessorfrequenz 286 geeignet und ermöglichten Geschwindigkeiten von bis zu 16 Megabit / Sekunde. Mit der Zeit wurde die von einem ISA-Bus angebotene Geschwindigkeit ungenügend. Obwohl theoretisch Geschwindigkeiten von bis zu 16

mult mai mici. A fost folosită cu succes pentru toate tipurile de controlere, însă s-a bucurat de succes până la sfârșitul anilor '90 pentru conectarea la calculator mai ales a controlerelor de rețea de până la 10 Mbps, a plăcilor de sunet și a modemurilor.

Calculatoarele personale de astăzi, păstrează încă o relică a acestui tip de magistrală. Portul pentru tastatură și mouse, iar în unele cazuri porturile seriale și paralele, precum și controlerul unității de dischetă, sunt conectate la o magistrală ISA care este cascădată la magistrala de I/O a sistemului prin intermediul magistralei PCI.

Megabyte / Sekunde auf einem ISA-Bus erreicht werden können, sind die tatsächlichen Geschwindigkeiten viel geringer. Es wurde erfolgreich für alle Arten von Controllern eingesetzt, war jedoch bis Ende der neunziger Jahre erfolgreich für den Anschluss von Netzwerkcontrollern mit bis zu 10 Mbit / s, insbesondere an den Computer Soundkarten und Modems.

Die heutigen PCs haben noch ein Relikt dieser Art von Bus. Der Tastatur- und Mausanschluss sowie in einigen Fällen der serielle und der parallele Anschluss sowie der Diskettenlaufwerk-Controller sind an einen ISA-Bus angeschlossen, der über den PCI-Bus mit dem System-E/A-Bus kaskadiert ist.

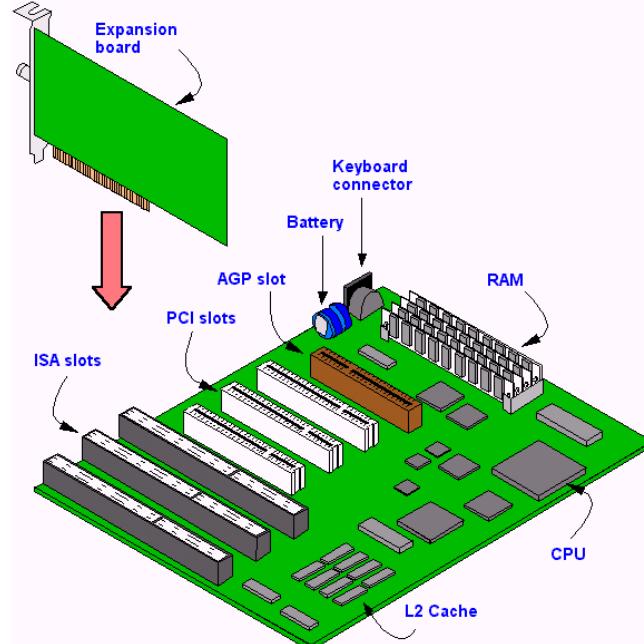


Figura 10. Conectori de magistrale (Busverbinder): ISA, PCI, AGP

4.2.2 Magistrala PCI (Peripheral Component Interconnect)

A fost introdusă de Intel la sfârșitul anilor 1990. Este pe 32 de biți și operează la frecvențe începând cu 33 MHz. Aceste valori permit viteze teoretice de până la 133 Megaocteți pe secundă. Majoritatea controlerelor existente în momentul de față

4.2.2 Der PCI-Bus (Peripheral Component Interconnect)

Es wurde Ende der neunziger Jahre von Intel eingeführt, ist 32-Bit-fähig und arbeitet mit Frequenzen ab 33 MHz. Diese Werte ermöglichen theoretische Geschwindigkeiten von bis zu 133 Megabyte pro Sekunde. Die meisten der derzeit vorhandenen Controller

se conectează la această magistrală: controlere IDE/ATA pentru conectarea hard disk-urilor, controlere de rețea, controlere multimedia de captură, sunet, etc. Este specifică și altor tipuri de calculatoare, nu numai calculatoarelor personale.

4.2.3 Magistrala AGP

S-a născut din nevoie de lățime de bandă mai mare spre controlerul video. Aplicații precum jocurile cereau o lățime de bandă pe care magistrala PCI nu o putea oferi.

De fapt numele de magistrala AGP este folosit impropriu, AGP fiind un canal de comunicație punct la punct. Magistrala AGP 1x este pe 32 de biți și lucrează la frecvențe de 66 MHz, oferind o viteză de aproximativ 266 Megaocetăi pe secundă – dublu față de viteza oferită de magistrala PCI. Această lățime de bandă este dedicată în întregime controlerului video. Magistralele AGP 8x actuale, permit atingerea unor viteze de 8 ori mai mari decât specificațiile AGP inițiale (2133 Megaocetăi pe secundă).

4.2.4 Magistrala PCI Express

Magistrala PCI Express (PCIe) reprezintă a treia generație a magistralei PCI, cu o performanță și fiabilitate mai ridicate comparativ cu generațiile anterioare PCI și PCI-X. Spre deosebire de aceste versiuni anterioare, care sunt magistrale paralele, PCIe este o magistrală serială. Din cauza naturii sale seriale, magistrala PCIe are mai multe avantaje față de o magistrală paralelă: număr mai redus de pini ai circuitelor integrate, complexitate mai redusă a plăcilor de circuite imprimate și cost mai redus al acestora. Numele PCI Express reflectă atât viteza ridicată a magistralei, cât și compatibilitatea software a acesteia cu generațiile anterioare PCI și

sind an diesen Bus verbunden: IDE / ATA-Controller zum Anschließen von Festplatten, Netzwerkcontrollern, Multimedia-Capture-Controllern, Sound usw. Es ist spezifisch für andere Computertypen, nicht nur für PCs.

4.2.3 Der AGP Bus

Es entstand aus der Notwendigkeit einer größeren Bandbreite für den Videocontroller. Anwendungen wie Spiele erforderten eine Bandbreite, die der PCI-Bus nicht bereitstellen konnte.

Tatsächlich wird der Name des AGP-Busses nicht richtig verwendet, da der AGP ein Punkt-zu-Punkt-Kommunikationskanal ist. Der AGP 1x-Bus ist 32-Bit und arbeitet mit Frequenzen von 66 MHz. Er bietet eine Geschwindigkeit von ca. 266 Megabyte pro Sekunde – doppelt so schnell wie der PCI-Bus. Diese Bandbreite ist ausschließlich dem Videocontroller vorbehalten. Die aktuellen 8x AGP-Busse ermöglichen Geschwindigkeiten, die bis zu achtmal höher sind als die ursprünglichen AGP-Spezifikationen (2133 Megabyte pro Sekunde).

4.2.4 Der PCI Express Bus

Der PCI Express-Bus (PCIe) ist die dritte Generation des PCI-Busses mit einer höheren Leistung und Zuverlässigkeit im Vergleich zu den vorherigen PCI- und PCI-X-Generationen. Im Gegensatz zu früheren Versionen, bei denen es sich um parallele Busse handelt, PCIe einen seriellen Bus ist. Aufgrund seiner seriellen Natur hat der PCIe-Bus mehrere Vorteile gegenüber einem parallelen Bus: weniger Pins von integrierten Schaltkreisen, geringere Komplexität von Leiterplatten und geringere Kosten. Der Name PCI Express widerspiegelt sowohl die hohe Geschwindigkeit des Busses als auch die Softwarekompatibilität mit früheren PCI-

PCI-X.

Proiectanții magistralei PCIe au menținut principalele caracteristici avantajoase ale arhitecturii generațiilor anterioare de magistrale PCI. De exemplu, ea utilizează același model de comunicație ca magistralele PCI și PCI-X. Sunt păstrate aceleași spații de adrese: de memorie, de I/E și de configurație. Magistrala PCIe permite utilizarea acelorași tipuri de tranzacții ca și magistralele anterioare: citire/scriere a spațiului de memorie, citire/scriere a spațiului de I/E, citire/scriere a spațiului de configurație. Prin aceasta, se păstrează compatibilitatea cu sistemele de operare și driverele software existente, care nu necesită modificări.

Ca și magistralele PCI anterioare, magistrala PCIe permite interconexiuni între circuite integrate și interconexiuni între plăci de circuite imprimate prin conectori și plăci de extensie. Plăcile de extensie au o structură similară cu plăcile utilizate de magistralele PCI și PCI-X. O placă de bază PCIe are dimensiuni similare cu plăcile de bază ATX existente, utilizate la calculatoarele personale. Pe lângă păstrarea unor caracteristici avantajoase ale magistralelor PCI și PCI-X, magistrala PCIe introduce diferite îmbunătățiri pentru creșterea performanței și reducerea costurilor.

Magistrala PCIe utilizează o interconexiune serială punct la punct pentru comunicația între două dispozitive periferice. În primul rând, o interconexiune serială elimină dezavantajele unei magistrale paralele, în special dificultatea sincronizării între liniile multiple de date datorită **nesimetriei de propagare a semnalelor**. Cauza acestei nesimetrii poate fi lungimea diferită a căilor de date parcuse de diferitele semnale sau propagarea pe straturi diferite ale plăcii de circuit imprimat. Deși semnalele de date ale

und PCI-X-Generationen.

Die Entwickler des PCIe-Busses haben die wichtigsten vorteilhaften Merkmale der Architektur früherer Generationen von PCI-Bussen beibehalten. Beispielsweise verwendet es das dasselbe Kommunikationsmodell wie die PCI- und PCI-X-Busse. Die gleichen Adressräume werden gespeichert: Speicher, E / A und Konfiguration. Mit dem PCIe-Bus können Sie dieselben Transaktionstypen wie mit früheren Bussen verwenden: Lese- / Schreibspeicher, Lese- / Schreib-E / A-Speicher, Lese- / Schreib-Konfigurationsspeicher. Auf diese Weise bleibt die Kompatibilität mit den vorhandenen Betriebssystemen und Softwaretreibern erhalten, für die keine Änderungen erforderlich sind.

Wie bei früheren PCI-Bussen ermöglicht der PCIe-Bus Verbindungen zwischen integrierten Schaltkreisen und Verbindungen zwischen Leiterplatten über Steckverbinder und Erweiterungskarten. Erweiterungskarten haben eine ähnliche Struktur wie PCI- und PCI-X-Busse. Ein PCIe-Motherboard hat eine ähnliche Größe wie die vorhandenen ATX-Motherboards, die auf PCs verwendet werden. Der PCIe-Bus behält nicht nur die vorteilhaften Eigenschaften von PCI- und PCI-X-Bussen bei, sondern führt auch verschiedene Verbesserungen ein, um die Leistung zu steigern und die Kosten zu senken.

Der PCIe-Bus verwendet eine serielle Punkt-zu-Punkt-Verbindung für die Kommunikation zwischen zwei Peripheriegeräten. Erstens beseitigt eine serielle Verbindung die Nachteile eines parallelen Busses, insbesondere die Schwierigkeit der Synchronisation zwischen mehreren Datenleitungen aufgrund der **Asymmetrie der Signalausbreitung**. Die Ursache für diese Asymmetrie kann die unterschiedliche Länge der Datenpfade sein, die von den unterschiedlichen Signalen

unei magistrale paralele sunt transmise simultan, ele pot ajunge la destinație la momente de timp diferite.

Creșterea frecvenței semnalului de ceas al unei magistrale paralele este dificilă, deoarece durata ciclului de ceas poate deveni mai mică decât durata nesimetriei de propagare a semnalelor (care poate fi de câteva nanosecunde). La o magistrală serială nu apare problema nesimetriei de propagare a semnalelor, deoarece nu există un semnal de ceas extern, informațiile de sincronizare fiind înglobate în semnalul serial transmis. În al doilea rând, o interconexiune punct la punct implică o încărcare electrică redusă a legăturii, ceea ce permite creșterea frecvenței semnalului de ceas utilizat pentru transferurile de date.

Magistrala PCIe dispune de mai multe facilități avansate. Astfel, facilitatea de calitate a serviciilor (Quality of Service – QoS) permite asigurarea unor performanțe diferențiate pentru diferite aplicații. Facilitatea de conectare și deconectare a unor module în timpul funcționării permite realizarea unor sisteme care funcționează fără întrerupere. Facilitățile de gestiune avansată a puterii + permit implementarea unor aplicații mobile cu un consum redus de putere. Facilitatea de gestiune a erorilor permite utilizarea magistralei PCIe în sistemele robuste necesare serverelor performante.

Principalele caracteristici ale magistralei PCIe sunt următoarele:

- Unifică arhitectura de I/E pentru diferite tipuri de sisteme (calculatoare de birou, calculatoare mobile, stații de lucru, servere, platforme de comunicație și sisteme înglobate)
- Permite interconectarea atât a circuitelor integrate de pe placă de bază,

zurückgelegt werden, oder die Ausbreitung auf unterschiedlichen Schichten der Leiterplatte. Die Datensignale eines Parallelbusses werden zwar gleichzeitig übertragen, können aber zu unterschiedlichen Zeiten das Ziel erreichen.

Das Erhöhen der Frequenz des Taktsignals eines Parallelbusses ist schwierig, da die Dauer des Taktzyklus kürzer werden kann als die Dauer der Signalausbreitungsasymmetrie (die einige Nanosekunden betragen kann). In einem seriellen Bus gibt es kein Problem der Asymmetrie der Signalausbreitung, da es kein externes Taktsignal gibt, wobei die Synchronisationsinformation in dem übertragenen seriellen Signal enthalten ist. Zweitens impliziert eine Punkt-zu-Punkt-Verbindung eine verringerte elektrische Ladung der Verbindung, wodurch die Frequenz des für Datenübertragungen verwendeten Taktsignals erhöht werden kann.

Der PCIe-Bus verfügt über mehrere erweiterte Funktionen. Somit ermöglicht die *Quality of Service* (QoS) -Funktion differenzierte Leistungen für verschiedene Anwendungen. Die einfache Verbindung und Trennung von Modulen während des Betriebs ermöglicht die Realisierung von Systemen, die ohne Unterbrechung arbeiten. Erweiterte Energieverwaltungsfunktionen ermöglichen die Bereitstellung mobiler Anwendungen mit geringem Energieverbrauch. Die Fehlermanagementfunktion ermöglicht die Verwendung des PCIe-Busses in den robusten Systemen, die für Hochleistungsserver erforderlich sind.

Die Hauptmerkmale des PCIe-Busses sind:

- Vereinheitlicht die E / A-Architektur für verschiedene Systemtypen (Bürocomputer, mobile Computer, Workstations, Server, Kommunikationsplattformen und eingebettete Systeme)
- Ermöglicht die Verbindung sowohl der integrierten Schaltkreise auf der

cât și a plăcilor de extensie prin intermediul unor conectori sau cabluri.

- Comunicația este bazată pe pachete, cu rată de transfer și eficiență ridicate.
- Interfața este serială, permitând reducerea numărului de pini și simplificarea conexiunilor.
- Performanța este scalabilă (o interconexiune poate fi implementată cu ajutorul mai multor benzi de comunicație).
- Modelul software este compatibil cu arhitectura PCI clasică (permite configurarea circuitelor PCIe, încărcarea sistemelor de operare și utilizarea driverelor software existente, fără a fi necesare modificări).
- Permite o calitate diferențiată a serviciilor (QoS).
- Pune la dispoziție o gestiune avansată a puterii consumate
- Asigură integritatea datelor la nivelul legăturii pentru toate tipurile de tranzacții.
- Permite raportarea și gestionarea avansată a erorilor pentru îmbunătățirea izolării defectelor și corectarea erorilor.
- Permite conectarea și deconectarea perifericelor în timpul funcționării, fără a fi necesară utilizarea unor semnale suplimentare.

Ansamblul format din controler, indiferent de tipul de magistrală I/O internă la care este conectat acesta, împreună cu interfața dintre acesta și echipamentul periferic este referit și sub numele de *magistrală externă*. Printre cele mai cunoscute magistrale externe amintim: IDE/ATA, SCSI, USB.

Hauptplatine als auch der Erweiterungsplatten über Steckverbinder oder Kabel.

- Die Kommunikation erfolgt paketbasiert mit hoher Übertragungsrate und Effizienz.
- Die Schnittstelle ist seriell, wodurch die Anzahl der Pins verringert und die Verbindungen vereinfacht werden.
- Die Performance ist skalierbar (eine Verschaltung kann mit Hilfe mehrerer Kommunikationsbänder realisiert werden).
- Das Softwaremodell ist mit der klassischen PCI-Architektur kompatibel (es ermöglicht die Konfiguration von PCIe-Schaltkreisen, das Laden von Betriebssystemen und die Verwendung vorhandener Softwaretreiber, ohne dass Änderungen erforderlich sind).
- Ermöglicht eine differenzierte Servicequalität (QoS).
- Bietet erweiterte Verwaltung des Stromverbrauchs
- Gewährleistet die Datenintegrität auf Verbindungsebene für alle Arten von Transaktionen.
- Ermöglicht erweiterte Fehlerberichte und -verwaltung, um die Fehlerisolierung und -korrektur zu verbessern.
- Ermöglicht das Anschließen und Trennen von Peripheriegeräten während des Betriebs, ohne dass zusätzliche Signale erforderlich sind.

Die Baugruppe bestehend aus der Steuerung, unabhängig vom Typ des internen E / A-Busses, an den sie angeschlossen ist, und der Schnittstelle zwischen ihr und den Peripheriegeräten wird auch als *externer Bus* bezeichnet. Zu den beliebtesten externen Bussen zählen: IDE / ATA, SCSI, USB.

4.3 Magistrale I/O externe și interfețe asociate

4.3.1 IDE / ATA (*Integrated Drive Electronics / Advanced Technology Attachment*)

Interfața IDE sau ATA a fost folosită de la mijlocul anilor '80 în vederea conectării perifericelor de stocare cum ar fi hard disk-urile și unitățile optice. Primele specificații ATA foloseau procesorul (așa numitul *programmed input/output mode* sau mod PIO) pentru a transfera informația octet cu octet de pe dispozitivul periferic de stocare în memoria internă prin intermediul registrelor procesorului. Acest mod de lucru permitea obținerea de viteze relativ mici de până la 16.7 Megaoceteți pe secundă și, mai grav, rețineau procesorul ocupat în timpul acestui transfer. Această viteză a crescut însă odată cu folosirea DMA (*Direct Memory Access*), permitându-se astfel copierea informației în blocuri de pe dispozitivul de stocare direct în memoria internă fără a mai implica procesorul în acest transfer.

Vitezele actuale ale interfețelor ATA 133 se apropie cel puțin teoretic de limitele magistralei PCI de 132 Megaoceteți pe secundă pentru transferul datelor (să nu uităm că informația citită de pe dispozitivul de stocare trebuie să treacă prin magistrala PCI via controller).

Specificațiile ATA descriu și modul de adresare a dispozitivului de stocare. Ultimele specificații prevăd o adresare pe 48 de biți, lucru care permite folosirea teoretică a dispozitivelor de stocare la capacitate de până la 32 de Teraocteți. Pe un controller IDE/ATA se pot conecta doar două dispozitive de stocare, unul operând în mod master și celălalt în mod slave. Majoritatea calculatoarelor

4.3 E/A Busse und zugehörige Schnittstellen

4.3.1 IDE / ATA (*Integrated Drive Electronics / Advanced Technology Attachment*)

Die IDE- oder ATA-Schnittstelle wird seit Mitte der 1980er Jahre zum Anschließen von Speicherperipheriegeräten wie Festplatten und optischen Laufwerken verwendet. Die ersten ATA-Spezifikationen verwendeten den Prozessor (den sogenannten *programmed input/output mode* oder den PIO-Modus), um die Byte-Byte-Informationen vom Peripheriespeichergerät über die Prozessorregister in den internen Speicher zu übertragen. Diese Betriebsart ermöglichte relativ niedrige Geschwindigkeiten von bis zu 16,7 Megabyte pro Sekunde und hielt den Prozessor während dieser Übertragung schlummer noch beschäftigt. Diese Geschwindigkeit hat sich jedoch mit der Verwendung von *Direct Memory Access* (DMA) erhöht, wodurch die Blockinformationen direkt von der Speichervorrichtung in den internen Speicher kopiert werden können, ohne dass der Prozessor an dieser Übertragung beteiligt ist. Die aktuellen Geschwindigkeiten der ATA 133-Schnittstellen liegen theoretisch mindestens nahe an den PCI-Busgrenzen von 132 Megabit pro Sekunde für die Datenübertragung (denken Sie daran, dass die vom Speichergerät gelesenen Informationen über den Controller über den PCI-Bus übertragen werden müssen). Die ATA-Spezifikationen beschreiben auch, wie das Speichergerät adressiert wird. Die neuesten Spezifikationen umfassen eine 48-Bit-Adresse, die die theoretische Verwendung von Speichergeräten mit einer Kapazität von bis zu 32 Terabyte ermöglicht. An einen IDE / ATA-Controller können nur zwei Speichergeräte angeschlossen werden, eines im Master-Modus und das andere im

personale, datorită faptului că sunt dotate cu două controlere IDE (numite primary și secondary IDE), pot folosi la un moment dat doar 4 dispozitive periferice IDE. Dacă se dorește folosirea unui număr mai mare de periferice IDE/ATA, este necesară instalarea unui controller IDE/ATA suplimentar, de obicei PCI.

4.3.2 SCSI (Small Computer System Interface)

Acest tip de magistrală s-a născut la mijlocul anilor '80, fiind cel mai des folosită pentru conectarea dispozitivelor de stocare cum ar fi hard disk-uri, unități optice, unități de bandă magnetică. Poate fi folosită și pentru conectarea altor tipuri de periferice (scannere, imprimante).

Poate fi folosită atât ca magistrală internă, cât și ca magistrală externă. În calculatoarele personale, a fost folosită doar ca magistrală externă, comunicând cu procesorul doar prin intermediul unui controller ISA sau PCI prin magistrala internă corespunzătoare tipului de controller. Funcțional, magistrala SCSI operează la frecvențe cuprinse între 5 MHz (conform primelor specificații SCSI) și 80 de MHz, dimensiunea magistralei fiind de 8 sau 16 biți. O magistrală SCSI pe 16 biți operând la 80 MHz poate atinge viteze de până la 320 Megaocteți pe secundă.

Fiecare dispozitiv periferic conectat la o magistrală SCSI își se asociază un identificator *SCSIid*. Numărul de biți pe care se reprezintă acest identificator implică și numărul de dispozitive care se pot conecta pe aceeași magistrală. Astfel, spre deosebire de IDE care permite doar două dispozitive periferice pe controller (magistrală), o magistrală SCSI poate suporta până la 8 sau 16 echipamente periferice. Un alt avantaj al controllerelor SCSI este că suportă *hot-swapping (hot-plugging)* – schimbarea în timpul mersului

Slave-Modus. Die meisten PCs können, da sie mit zwei IDE-Controllern (primäre und sekundäre IDE) ausgestattet sind, nur 4 IDE-Peripheriegeräte gleichzeitig verwenden. Wenn Sie mehr IDE / ATA-Peripheriegeräte verwenden möchten, müssen Sie einen zusätzlichen IDE / ATA-Controller installieren, in der Regel PCI.

4.3.2 SCSI (Small Computer System Interface)

Dieser Bustyp wurde Mitte der 1980er Jahre geboren und wird am häufigsten zum Anschließen von Speichergeräten wie Festplatten, optischen Laufwerken und Magnetbandlaufwerken verwendet. Es kann auch zum Anschließen anderer Peripheriegeräte (Scanner, Drucker) verwendet werden.

Es kann sowohl als interner Bus als auch als externer Bus verwendet werden. In Personalcomputern wurde er nur als externer Bus verwendet und kommunizierte mit dem Prozessor nur über einen ISA- oder PCI-Controller über den internen Bus, der dem Controllertyp entspricht. Funktionell arbeitet der SCSI-Bus mit Frequenzen zwischen 5 MHz (gemäß der ersten SCSI-Spezifikation) und 80 MHz, wobei die Busgröße 8 oder 16 Bit beträgt. Ein 16-Bit-SCSI-Bus mit 80 MHz kann Geschwindigkeiten von bis zu 320 Megabit pro Sekunde erreichen.

Jedem an einen SCSI-Bus angeschlossenen Peripheriegerät wird eine *SCSIid*-Kennung zugewiesen. Die Anzahl der Bits, die durch diesen Bezeichner dargestellt werden, impliziert auch die Anzahl der Geräte, die an denselben Bus angeschlossen werden können. Im Gegensatz zu IDE, die nur zwei Peripheriegeräte pro Controller (Bus) zulässt, kann ein SCSI-Bus somit bis zu 8 oder 16 Peripheriegeräte unterstützen. Ein weiterer Vorteil von SCSI-Controllern besteht darin, dass sie *hot-swapping (hot-Plugging)* unterstützen – das Umschalten

calculatorului a echipamentelor periferice conectate. Interfața serială comunică serial cu un dispozitiv periferic, transferul de date făcându-se pe principiul serial, bit cu bit. Cele mai des întâlnite periferice seriale sunt mouse-urile, modemurile și terminalele virtuale.

4.3.3 Interfața serială

Ea comunică serial cu un dispozitiv periferic, transferul de date făcându-se pe principiul serial, bit cu bit. Cele mai des întâlnite periferice seriale sunt mouse-urile, modemurile și terminalele virtuale.

Controlerul serial la calculatoarele personale de azi este pe cale de dispariție, majoritatea perifericelor care se conectau la calculator pe această interfață conectându-se în prezent prin USB. Acolo unde este încă prezent, controlerul serial este legat de magistrala ISA sau PCI. Viteza maximă atinsă pe un port serial este foarte mică, 128000 biți /secundă \approx 16 kiloocteți / secundă.

4.3.4 Interfața paralelă

Este și ea o relicvă în calculatoarele moderne, încet renunțându-se la ea și datorită nevoii de a reduce costurile unui sistem de calcul și mai ales datorită numărului mic de periferice care mai folosesc acest tip de interfață. Principiul de comunicare pe o asemenea interfață este bineînțeles cel paralel – mai mulți biți sunt transferați în același timp pe mai multe fire fizice.

În prezent, singurele echipamente întâlnite care folosesc această interfață sunt imprimantele. În trecut, interfața paralelă a fost folosită și pentru conectarea altor

angeschlossener Peripheriegeräte während des Betriebs des Computers. Die serielle Schnittstelle kommuniziert seriell mit einem Peripheriegerät, wobei die Datenübertragung nach dem seriellen Prinzip Bit für Bit erfolgt. Die gebräuchlichsten seriellen Peripheriegeräte sind Mäuse, Modems und virtuelle Terminals.

4.3.3 Serielle Schnittstelle

Sie kommuniziert seriell mit einem Peripheriegerät, wobei die Datenübertragung nach dem seriellen Prinzip Bit für Bit erfolgt. Die gebräuchlichsten seriellen Peripheriegeräte sind Mäuse, Modems und virtuelle Terminals.

Der serielle Controller heutiger PCs ist vom Aussterben bedroht. Die meisten Peripheriegeräte, die über diese Schnittstelle an den Computer angeschlossen sind, sind derzeit über USB angeschlossen. Wo es noch vorhanden ist, wird der serielle Controller an den ISA- oder PCI-Bus angeschlossen. Die maximale Geschwindigkeit, die an einem seriellen Port erreicht wird, ist sehr niedrig: 128000 Bit / Sekunde \approx 16 Kilobyte / Sekunde.

4.3.4 Parallele Schnittstelle

Es ist auch ein Relikt in modernen Computern, das langsam aufgibt und aufgrund der Notwendigkeit, die Kosten eines Computersystems zu senken, und insbesondere aufgrund der geringen Anzahl von Peripheriegeräten, die diese Art von Schnittstelle verwenden. Das Kommunikationsprinzip auf einer solchen Schnittstelle ist natürlich das parallele – viele Bits werden gleichzeitig auf mehreren physischen Drähten übertragen.

Derzeit sind die einzigen Geräte, die diese Schnittstelle verwenden, Drucker. In der Vergangenheit wurde die parallele Schnittstelle auch zum Anschluss anderer

periferice cum ar fi camerele video sau scannerele. Producătorii de asemenea echipamente periferice au renunțat la interfața paralelă în favoarea celei USB. Ca și în cazul controlerului serial, unde mai este prezent, controlerul paralel este legat de magistrala ISA sau PCI.

4.3.5 USB (Universal Serial Bus)

Interfața USB s-a născut din necesitatea unui mecanism universal de conectare a echipamentelor periferice la calculator, indiferent de tipul acestora. Rata de transfer merge de la 1.5 Mbit/secundă (USB 1.0) la 10 Gbit/secundă (USB 3.1 Gen 2), suficientă pentru conectarea oricărui dispozitiv extern cu calculatorul, mai puțin a unui monitor. Prințipiu de comunicare cu acestea este serial, interfața USB folosind pentru date doar două fire. Pe un singur + USB se pot conecta până la 127 de echipamente USB. Fiecare echipament primind un USB id reprezentat pe 7 biți (de fapt un id este consumat de controlerul USB însuși).

Specificațiile USB permit adăugarea echipamentelor periferice USB sau înlocuirea acestora chiar în timpul funcționării calculatorului.

5. Arhitectura microprocesoarelor x86 (IA-32)

5.1 Structura microprocesorului

Microprocesorul x86 este format din două componente principale:

- **EU (Executive Unit)** – execuță instrucțiunile mașină prin intermediul componentei ALU.
- **BIU (Bus Interface Unit)** – pregătește execuția fiecărei instrucțiuni mașină. Citește o instrucțiune din memorie, o

Peripheriegeräte wie Camcorder oder Scanner verwendet. Hersteller solcher Peripheriegeräte haben auch die parallele Schnittstelle zugunsten von USB aufgegeben. Wie beim seriellen Controller, wo er noch vorhanden ist, ist der parallele Controller mit dem ISA- oder PCI-Bus verbunden.

4.3.5 USB (Universal Serial Bus)

Die USB-Schnittstelle entstand aus der Notwendigkeit eines universellen Mechanismus zum Anschließen von Peripheriegeräten an den Computer, unabhängig von ihrem Typ. Die Übertragungsrate reicht von 1,5 Mbit / s (USB 1.0) bis 10 Gbit / s (USB 3.1 Gen 2), um alle externen Geräte mit Ausnahme eines Monitors an den Computer anzuschließen. Das Prinzip der Kommunikation mit diesen ist seriell, wobei die USB-Schnittstelle für Daten nur zwei Drähte verwendet. Bis zu 127 USB-Geräte können an einen einzelnen USB-Controller angeschlossen werden. Jedes Gerät erhält eine 7-Bit-USB-ID (tatsächlich wird eine ID vom USB-Controller selbst verwendet).

USB-Spezifikationen ermöglichen das Hinzufügen oder Ersetzen von USB-Peripheriegeräten, auch wenn der Computer ausgeführt wird.

5. Architektur von x86-Mikroprozessoren (IA-32)

5.1 Aufbau des Mikroprozessors

Der x86-Mikroprozessor besteht aus zwei Hauptkomponenten:

- **EU (Executive Unit)** – führt die Maschinenanweisungen über die ALU-Komponente aus.
- **BIU (Bus Interface Unit)** – bereitet die Ausführung jeder MaschinenAnweisung vor. Liest einen Anweisung aus dem

decodifică și calculează adresa din memorie a unui eventual operand. Configurația rezultată este depusă într-o zonă tampon cu dimensiunea de 15 octeți, de unde va fi preluată de EU.

EU și BIU lucrează în paralel – în timp ce EU execută instrucțiunea curentă, BIU pregătește instrucțiunea următoare. Cele două acțiuni sunt sincronizate – cea care termină prima așteaptă după cealaltă.

Speicher, decodiert ihn und berechnet die Speicheradresse eines möglichen Operanden. Die resultierende Konfiguration wird in einer Pufferzone mit einer Größe von 15 Byte gespeichert und von der EU übernommen.

EU und BIU arbeiten parallel – während die EU die aktuelle Anweisung ausführt, bereitet die BIU die nächste Anweisung vor. Diesen Aktionen sind synchronisiert – die eine, die die erste beendet, wartet nach der anderen.

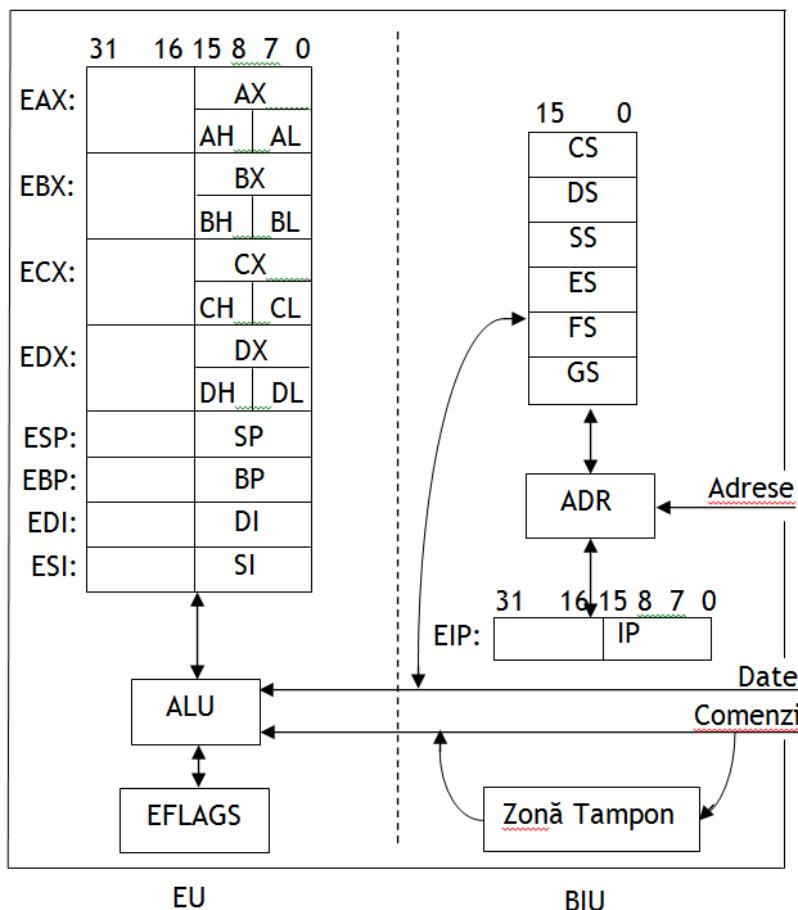


Figura 11. Arhitectura microprocesorului 8086 (Architektur von 8086-Mikroprozessor)

5.1.1 Reștricții generali ai EU

- 1) **Registrul EAX (Extended AX)** este registrul acumulator. El este folosit de către majoritatea instrucțiunilor ca unul dintre operanzi;
- 2) **Registrul EBX** – registru general;

5.1.1 EU-Allgemeine Register

- 1) Das **EAX-Register (Extended AX)** ist das Akumulatorregister. Es wird von den meisten Anweisungen als einer der Operanden verwendet;
- 2) **EBX-Register** - Allgemeines Register;

- 3) **Registrul ECX** – regisztr de numărare (regiszru contor) pentru instrucțiunile care au nevoie de indicații numerice.
- 4) **Registrul EDX** – regisztr de date. Împreună cu EAX se folosește în calculele ale căror rezultate depășesc un dublucuvânt (32 biți).

Un „cuvânt binar” (*binary word*) este o reprezentare a unui număr într-un sistem de numerație binar pe un anumit număr de biți dat. „Dimensiunea cuvântului” se referă la numărul de biți procesați de CPU-ul unui SC la un moment dat (de regulă, 32 de biți sau 64 de biți). Dimensiunea sau lățimea de bandă a magistralelor de date sau de adrese, dimensiunea instrucțiunilor etc. sunt de regulă multipli ai acestui număr.

Pentru a complica lucrurile, pentru sănumita *backwards compatibility* (compatibilitate retroactivă), Microsoft Windows API definește un WORD ca fiind un TIP DE DATE pe 16 biți, un DWORD ca fiind un TIP DE DATE pe 32 de biți și un QWORD pe 64 de biți, indiferent de procesor.

5) **Regiștrii ESP și EBP** sunt regiștri destinați lucrului cu stiva. O stivă se definește ca fiind o zonă de memorie în care se pot depune succesiv valori, extragerea lor ulterioră făcându-se în ordinea inversă depunerii. Registrul ESP (*Stack Pointer*) punctează spre elementul ultim introdus în stivă (elementul din vârful stivei). Registrul EBP (*Base pointer*) punctează spre primul element introdus în stivă (indică baza stivei).

6) **Regiștrii EDI și ESI** (*Destination Index și Source Index*) sunt regiștri de index utilizati de obicei pentru accesarea elementelor din siruri de octeți sau de cuvinte.

Fiecare dintre regiștrii EAX, EBX, ECX,

- 3) **ECX-Register** – Zählregister für die Anweisungen, die numerische Informationen benötigen.
- 4) **EDX-Register** – Datenregister. Zusammen mit EAX wird es in Berechnungen verwendet, deren Ergebnisse ein Doppelwort (32 Bit) überschreiten.

Ein „binäres Wort“ ist eine Darstellung einer Zahl in einem binären Nummerierungssystem für eine gegebene Anzahl von Bits. „Wortgröße“ bezieht sich auf die Anzahl von Bits, die von der CPU eines CS zu einer gegebenen Zeit verarbeitet werden (normalerweise 32 Bits oder 64 Bits). Die Größe oder Bandbreite der Daten- oder Adressbusse, die Größe der Anweisungen usw. in der Regel sind Vielfaches dieser Zahl.

Um die Abwärtskompatibilität (*backwards compatibility*) zu gewährleisten, definiert die Microsoft Windows-API ein WORD als 16-Bit-Datentyp, ein DWORD als 32-Bit-Datentyp und ein 64-Bit-QWORD. Bits, unabhängig vom Prozessor.

5) **ESP- und EBP-Register** sind Register, die für die Stapelverarbeitung vorgesehen sind. Ein Stapel ist definiert als ein Speicherbereich, in dem Werte nacheinander abgelegt werden können, wobei ihre anschließende Entnahme in umgekehrter Reihenfolge der Ablage erfolgt. Das Register ESP (*Stack Pointer*) zeigt auf das zuletzt in den Stapel eingefügte Element (das Element oben im Stapel). Das EBP-Register (*Base Pointer*) zeigt auf das erste Element, das in den Stapel eingegeben wurde (gibt die Basis des Staples an).

6) **Die EDI und ESI Register** (*Destination Index und Source Index*) sind Indexregister, die üblicherweise für den Zugriff auf Elemente in Byte- oder Wortfolgen verwendet werden.

EDX, ESP, EBP, EDI, ESI au capacitatea de 32 biți. Fiecare dintre ei poate fi privit în același timp ca fiind format prin concatenarea (alipirea) a doi (sub)registri de câte 16 biți. Subregistru superior, care conține cei mai semnificativi 16 biți ai registrului de 32 biți din care face parte, nu are denumire și nu este disponibil separat. Subregistru inferior poate însă fi accesat individual, având astfel registrii de 16 biți AX, BX, CX, DX, SP, BP, DI, SI. Dintre aceștia, registrii AX, BX, CX, și DX sunt, fiecare la rândul lor, formați din câte doi alți subregistri a câte 8 biți. Există astfel registrii AH, BH, CH, DH, conținând cei 8 biți superiori (partea HIGH a registriilor AX, BX, CX și DX), respectiv AL, BL, CL, DL, conținând cei 8 biți inferiori (partea LOW).

Jedes der EAX-, EBX-, ECX-, EDX-, ESP-, EBP-, EDI- und ESI-Register hat eine Kapazität von 32 Bit. Jedes von ihnen kann gleichzeitig als durch Verketten (Anhängen) von zwei (Unter-) Registern mit jeweils 16 Bits gebildet angesehen werden. Das obere Unterregister, das die höchstwertigen 16 Bits des 32-Bit-Registers enthält, zu dem es gehört, hat keinen Namen und ist nicht separat verfügbar. Auf den unteren Teilbereich kann jedoch individuell zugegriffen werden, so dass die 16-Bit-Register AX, BX, CX, DX, SP, BP, DI, SI vorliegen. Davon bestehen die Register AX, BX, CX und DX jeweils aus zwei weiteren 8-Bit-Unterregistern. Es gibt also Register AH, BH, CH, DH, die die 8 oberen Bits enthalten (der HIGH-Teil der Register AX, BX, CX und DX) bzw. AL, BL, CL, DL, die die 8 unteren Bits enthalten (der LOW-Teil).

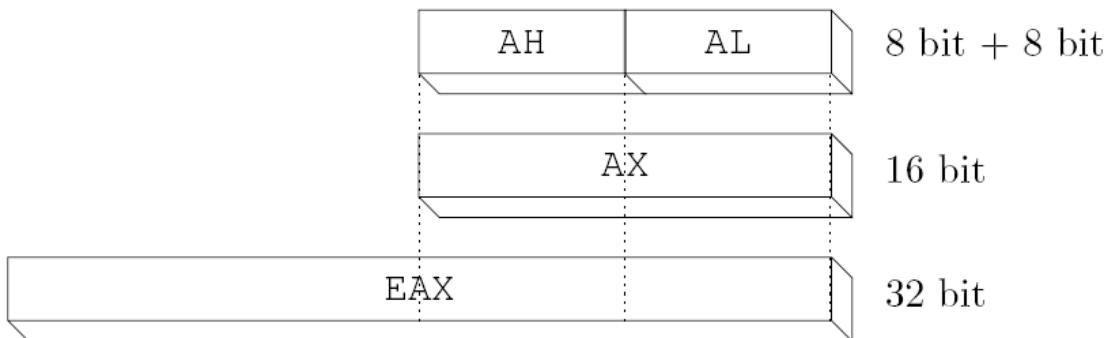


Figura 12. Structura registrului EAX (Struktur des EAX-Registers)

5.1.2 Flagurile

Un *flag* este un indicator reprezentat pe un bit. O configurație a *registrului de flag-uri* indică un rezumat sintetic al execuției fiecărei instrucțiuni. Pentru x86 registrul EFLAGS (the *status register*) are 32 biți dintre care sunt folosiți ușual numai 9.

5.1.2 Flaggen

Ein Flag ist ein Indikator, der auf einem Bit dargestellt wird. Eine *Flagregisterkonfiguration* gibt eine Zusammenfassung der Ausführung jeder Anweisung an. Für x86 hat das EFLAGS-Register (das *Statusregister*) 32 Bits, von denen nur 9 gemeinsam verwendet werden.

31	30	...	12	11	10	9	8	7	6	5	4	3	2	1	0
x	x	...	x	OF	DF	IF	TF	SF	ZF	x	AF	x	PF	x	CF

CF (*Carry Flag*) este *flag*-ul de transport. Are valoarea 1 în cazul în care în cadrul ultimei operații efectuate (UOE) s-a efectuat transport în afara domeniului de reprezentare a rezultatului și valoarea 0 în caz contrar. De exemplu, pentru:

$$\begin{array}{r} 1001\ 0011 \\ \underline{0111\ 0011} \\ \hline \textcolor{red}{1}0000\ 0110 \end{array} \quad + \quad \begin{array}{r} 146 \\ \underline{115} \\ \hline \textcolor{red}{0}262 \end{array}$$

rezultă un transport de cifră semnificativă și valoarea 1 este depusă automat în CF.

Flagul CF semnalează depășirea în cazul interpretării FĂRĂ SEMN.

PF (*Parity Flag*) – Valoarea lui se stabilește astfel încât împreună cu numărul de biți de 1 din octetul cel mai puțin semnificativ al reprezentării rezultatului UOE să rezulte un număr impar de cifre.

AF (*Auxiliary Flag*) indică valoarea transportului de la bitul 3 la bitul 4 al rezultatului UOE. De exemplu, în adunarea de mai sus bitul de transport este 0.

ZF (*Zero Flag*) primește valoarea 1 dacă rezultatul UOE este egal cu zero și valoarea 0 la rezultat diferit de zero.

SF (*Sign Flag*) primește valoarea 1 dacă rezultatul UOE este un număr strict negativ și valoarea 0 în caz contrar.

TF (*Trap Flag*) este un *flag* de depanare. Dacă are valoarea 1, atunci mașina se oprește după fiecare instrucțiune.

IF (*Interrupt Flag*) este *flag* de întrerupere. Asupra acestui *flag* vom reveni.

DF (*Direction Flag*) – pentru operarea asupra sirurilor de octeți sau de cuvinte.

CF (*Carry Flag*) ist die Transportflagge. Es hat den Wert 1, wenn in der zuletzt ausgeführten Operation (ZAO) außerhalb des Darstellungsfeldes des Ergebnisses transportiert wurde und den Wert 0, ansonsten. Zum Beispiel für:

$$\begin{array}{r} 93h \\ \underline{73h} \\ \hline \textcolor{red}{1}06h \end{array} \quad + \quad \begin{array}{r} -109 \\ \underline{115} \\ \hline 06 \end{array}$$

es wird eine signifikante Zahl transportiert und der Wert 1 wird automatisch im CF abgelegt.

Das CF-Flag signalisiert die Überwindung bei Interpretation OHNE ZEICHEN.

PF (*Parity Flag*) – Sein Wert wird so eingestellt, dass sich zusammen mit der Anzahl der Bits von 1 des niedrigstwertigen Bytes der Darstellung des ZAO-Ergebnisses eine ungerade Anzahl von Ziffern 1 ergibt.

AF (*Auxiliary Flag*) zeigt den Transportwert von Bit 3 bis Bit 4 des ZAO-Ergebnisses an. In der obigen Versammlung ist der Transportbit beispielsweise 0.

ZF (*Zero Flag*) erhält den Wert 1, wenn das Ergebnis der ZAO gleich Null ist und der Wert 0 am Ergebnis von Null abweicht.

SF (*Sign Flag*) erhält den Wert 1, wenn das ZAO-Ergebnis eine streng negative Zahl ist, und den Wert 0, wenn dies nicht der Fall ist.

TF (*Trap Flag*) ist ein Fehlerbehebungsflag. Wenn es auf 1 gesetzt ist, stoppt die Maschine nach jeder Anweisung.

IF (*Interrupt Flag*) ist das Interrupt-Flag. Auf diese Flagge werden wir zurückkommen.

DF (*Direction Flag*) – für den Betrieb an Byte- oder Wortketten. Wenn es den Wert 0

Dacă are valoarea 0, atunci deplasarea în sir se face de la început spre sfârșit, iar dacă are valoarea 1 este vorba de deplasări de la sfârșit spre început.

OF (*Overflow Flag*) este *flag* pentru depășire în cazul operațiilor asupra numerelor **CU SEMN**. Dacă rezultatul ultimei instrucțiuni în interpretarea CU SEMN a operanzilor nu a încăput în spațiul rezervat operanzilor (intervalul de reprezentare admisibil), atunci acest *flag* va avea valoarea 1, altfel va avea valoarea 0.

5.1.3 Regiștrii de adresă și calculul de adresă

Definiție

Adresa unei locații este numărul de octeți consecutivi dintre începutul memoriei RAM și începutul locației respective.

O succesiune continuă de locații de memorie, menite să deservească scopuri similare în timpul execuției unui program, formează un *segment*. În consecință, un segment reprezintă o diviziune logică a memoriei unui program, caracterizată prin *adresa de bază* (început), *limita* (dimensiunea) și *tipul* acesteia. Atât adresa de bază cât și dimensiunea unui segment au valori reprezentate pe 32 biți.

În familia procesoarelor bazate pe arhitectura 8086, termenul **segment** are două înțeleșuri:

1. Un bloc de memorie de dimensiune discretă, numit *segment fizic*. Numărul de octeți dintr-un segment de memorie fizică este de 64K pentru procesoarele pe 16 biți, respectiv 4 GB pentru procesoarele pe 32 de biți.
2. Un bloc de memorie de dimensiune variabilă, numit *segment logic*, ocupat de

hat, wird die Bewegung in der Zeichenfolge vom Anfang bis zum Ende ausgeführt, und wenn es den Wert 1 hat, handelt es sich um die Bewegungen vom Ende bis zum Anfang.

OF (*Overflow Flag*) ist das Flag, das bei Operationen mit **Vorzeichenzahle** überschritten werden soll. Wenn das Ergebnis der letzten Anweisung in der Vorzeichenzahle-Interpretation der Operanden nicht in den für die Operanden reservierten Bereich (zulässiger Darstellungsbereich) passte, hat dieses Flag den Wert 1, andernfalls den Wert 0.

5.1.3 Adressregister und Adressberechnung

Definition

Die Adresse eines Speicherorts ist die Anzahl aufeinanderfolgender **Bytes** zwischen dem Beginn des RAM und dem Beginn dieses Speicherorts.

Eine fortlaufende Folge von Speicherplätzen, die während der Programmausführung ähnlichen Zwecken dienen sollen, bildet ein *Segment*. Folglich stellt ein Segment eine logische Aufteilung des Programmspeichers dar, die durch seine *Basisadresse* (Start), *Grenze* (Größe) und seinen *Typ* gekennzeichnet ist. Sowohl die Basisadresse als auch die Größe eines Segments haben Werte, die auf 32 Bits dargestellt sind.

In der Familie der Prozessoren, die auf der 8086-Architektur basieren, hat der Begriff **Segment** zwei Bedeutungen:

1. Ein Speicherblock mit diskreter Größe, der als *physikalisches Segment* bezeichnet wird. Die Anzahl der Bytes in einem physischen Speichersegment beträgt 64 KB für 16-Bit-Prozessoren bzw. (beziehungsweise) 4 GB für 32-Bit-Prozessoren.
2. Ein Speicherblock mit variabler Größe, der als *logisches Segment* bezeichnet wird

codul sau datele unui program.

Vom numi *offset* sau *deplasament* adresa unei locații față de începutul unui segment, sau, cu alte cuvinte, numărul de octeți aflați între începutul segmentului și locația în cauză. Un *offset* se consideră valid dacă și numai dacă valoarea sa numerică, pe 32 biți, nu depășește limita segmentului la care se raportează.

Vom numi *specificare de adresă* (sau *adresă logică*) o pereche formată dintr-un *selector de segment* și un *offset*. Un **selector de segment** este o valoare numerică de 16 biți care identifică (indică/selectează) în mod unic segmentul accesat și caracteristicile acestuia. În scriere hexazecimală o adresă se exprimă sub forma:

s₃s₂s₁s₀ : 0706050403020100

În acest caz, selectorul *s₃s₂s₁s₀* indică accesarea unui segment a cărui adresă de bază este de forma $b_7b_6b_5b_4b_3b_2b_1b_0$ și având o limită $l_7l_6l_5l_4l_3l_2l_1l_0$. Baza și limita sunt determinate de către procesor în urma aplicării mecanismului de segmentare.

Pentru a fi permis accesul către locația specificată, este necesar să fie îndeplinită condiția:

0706050403020100 ≤ l₇l₆l₅l₄l₃l₂l₁l₀

Determinarea *adresei de segmentare* din specificarea de adresă se face conform formulei:

a₇a₆a₅a₄a₃a₂a₁a₀ := b₇b₆b₅b₄b₃b₂b₁b₀ + 0706050403020100

unde $a_7a_6a_5a_4a_3a_2a_1a_0$ este adresa calculată (scrisă în hexazecimal). Adresa rezultată din calculul de mai sus, poartă numele de *adresă liniară*.

O specificare de adresă mai poartă și numele de adresă FAR (îndepărtată). Atunci când o adresă se precizează doar prin *offset*, spunem ca este o adresă NEAR (apropiată), pentru că se subînțelege că

und mit dem Code oder den Daten eines Programms belegt ist.

Wir nennen *Offset* die Adresse eines Ortes gegenüber dem Beginn eines Segments, oder mit anderen Worten die Anzahl der Bytes zwischen dem Beginn des Segments und dem betreffenden Ort. Ein *Offset* gilt nur dann als gültig, wenn sein numerischer Wert auf 32 Bit die Segmentgrenze, auf die er sich bezieht, nicht überschreitet.

Wir werden die *Adressspezifikation* (oder *logische Adresse*) ein Paar nennen, das aus einem *Segmentselektor* und einem *Offset* besteht. Ein **Segmentselektor** ist ein numerischer 16-Bit-Wert, der das Segment, auf das zugegriffen wird, und seine Eigenschaften eindeutig identifiziert (anzeigt / auswählt). In **hexadezimaler** Schreibweise wird eine Adresse ausgedrückt als:

In diesem Fall zeigt der Selektor *s₃s₂s₁s₀* den Zugriff auf ein Segment an, dessen Basisadresse die Form $b_7b_6b_5b_4b_3b_2b_1b_0$ hat und dessen Grenze $l_7l_6l_5l_4l_3l_2l_1l_0$ ist. Die Basis und die Grenze werden vom Prozessor nach Anwendung des Segmentierungsmechanismus bestimmt. Um Zugriff auf den angegebenen Ort zu erhalten, muss die Bedingung erfüllt sein:

l₇l₆l₅l₄l₃l₂l₁l₀ ≤ l₇l₆l₅l₄l₃l₂l₁l₀

Die Ermittlung der Segmentierungsadresse aus der Adressangabe erfolgt nach der Formel:

a₇a₆a₅a₄a₃a₂a₁a₀ := b₇b₆b₅b₄b₃b₂b₁b₀ + 0706050403020100

dabei ist $a_7a_6a_5a_4a_3a_2a_1a_0$ die berechnete Adresse (hexadezimal geschrieben). Die aus der obigen Berechnung resultierende Adresse wird als lineare Adresse bezeichnet.

Eine Adressspezifikation trägt auch den FAR (*remote*) -Adressennamen. Wenn eine Adresse nur durch einen *Offset* angegeben wird, sagen wir, dass es sich um eine NEAR (schließen) -Adresse handelt, da davon

operăm în același segment.

Un exemplu concret de specificare de adresă este:

8:1000h

Pentru a calcula adresa liniară care corespunde acestei specificări, procesorul va proceda după cum urmează:

1. Verifică dacă segmentul care corespunde valorii de selector 8 a fost definit de către sistemul de operare. Dacă un astfel de segment nu a fost definit, se blochează accesul;
2. Extrage adresa de bază (B) și limita acestui segment (L), de exemplu, ca rezultat am putea avea $B = 2000h$ și $L = 4000h$;
3. Verifică dacă offset-ul depășește limita segmentului: $1000h > 4000h$? În caz de depășire, accesul ar fi fost blocat;
4. Adună offset-ul cu B, obținând în cazul nostru adresa liniară $3000h$ ($1000h + 2000h$). Acest calcul este efectuat de către componenta **ADR** din **BIU**.

Acest mecanism de adresare poartă numele de segmentare, vorbind astfel despre modelul de adresare segmentată.

În cazul în care segmentele încep la adresa 0 și au dimensiunea maximă posibilă (4GB), orice offset este automat valid și segmentarea nu contribuie efectiv în calculul adreselor. Astfel, având $b_7b_6b_5b_4b_3b_2b_1b_0 = 00000000$, calculul de adresă pentru adresa logică $s_3s_2s_1s_0:0706050403020100$ va rezulta în adresa liniară:

$$a_7a_6a_5a_4a_3a_2a_1a_0 := 00000000 + 0706050403020100$$

$$a_7a_6a_5a_4a_3a_2a_1a_0 := 0706050403020100$$

Acest mod particular de utilizare a segmentării, folosit de către majoritatea

ausgegangen wird, dass wir im selben Segment arbeiten.

Ein konkretes Beispiel für eine Adressangabe ist:

8:1000h

Um die dieser Spezifikation entsprechende lineare Adresse zu berechnen, geht der Prozessor wie folgt vor:

1. Überprüft, ob das Segment, das dem Auswahlwert 8 entspricht, vom Betriebssystem definiert wurde. Wenn ein solches Segment nicht definiert wurde, wird der Zugriff gesperrt.
2. Extrahiert die Basisadresse (B) und die Grenze dieses Segments (L). Als Ergebnis könnten $B = 2000h$ und $L = 4000h$ sein.
3. Überprüft, ob der Offset die Segmentgrenze überschreitet: $1000h > 4000h$? Im Falle einer Überwindung wäre der Zugang gesperrt worden;
4. Sammelt den Offset mit B und erhältet in unserem Fall die lineare Adresse $3000h$ ($1000h + 2000h$). Diese Berechnung wird von der ADR-Komponente der BIU durchgeführt.

Dieser Adressierungsmechanismus trägt den Segmentierungsnamen und spricht somit vom segmentierten Adressierungsmodell.

Wenn die Segmente bei Adresse 0 beginnen und die maximal mögliche Größe (4 GB) haben, ist jeder Offset automatisch gültig und die Segmentierung trägt nicht effektiv zur Berechnung der Adressen bei. Wenn also $b_7b_6b_5b_4b_3b_2b_1b_0 = 00000000$ ist, ergibt die Adressberechnung für die logische Adresse $s_3s_2s_1s_0:0706050403020100$ die lineare Adresse:

Diese spezielle Art der Verwendung der Segmentierung, die von den meisten

sistemelor de operare moderne poartă numele de *model de memorie flat*.

Procesoarele x86 suportă și un mecanism de control al accesului la memorie. Acesta se numește *paginare* și este independent de adresarea segmentată. Paginarea implică împărțirea **memoriei virtuale** în pagini, care sunt asociate (se spune că sunt translatate) memoriei fizice disponibile.

Configurarea și controlul mecanismelor de segmentare și de paginare sunt sarcina sistemului de operare. Dintre cele două, doar segmentarea intervine în specificarea de adrese, paginarea fiind complet **transparentă** din perspectiva programelor de utilizator („transparent” înseamnă că programele nu „văd” nimic cu privire la acest mecanism).

Atât calculul de adrese cât și folosirea mecanismelor de segmentare și paginare sunt influențate de *modul de execuție* al procesorului, procesoarele x86 suportând următoarele moduri de execuție mai importante:

1. *mod real*, pe 16 biți (folosind cuvânt de memorie de 16 biți și având memoria limitată la 1 MB);
2. ***mod protejat pe 16 sau 32 biți, caracterizat prin folosirea paginării și segmentării;***
3. *mod virtual 8086*, permite rularea programelor de tip mod real alături de cele de mod protejat;
4. *long mode*, pe 64 sau 32 biți, unde paginarea este obligatorie în timp ce segmentarea este dezactivată.

În cadrul cursului ne vom concentra asupra arhitecturii și comportamentului procesoarelor x86 în modul protejat pe 32 de biți.

Arhitectura x86 permite folosirea a patru tipuri de segmente cu roluri diferite:

modernen Betriebssystemen verwendet wird, wird als *flaches Speichermodell* bezeichnet. Die x86-Prozessoren unterstützen auch einen Mechanismus zur Speicherzugriffskontrolle. Dies wird als *Paging* bezeichnet und ist unabhängig von der segmentierten Adressierung. Bei einer Seite wird der **virtuelle Speicher** in *Seiten* aufgeteilt, die sind dem verfügbaren physischen Speicher zugeordnet (sie sollen übersetbar sein).

Das Konfigurieren und Steuern der Segmentierungs- und Paging-Mechanismen ist Aufgabe des Betriebssystems. Von den beiden ist nur die Segmentierung bei der Angabe von Adressen beteiligt, wobei die Seite aus Sicht von Benutzerprogrammen vollständig **transparent** ist („transparent“ bedeutet, dass die Programme nichts über diesen Mechanismus „sehen“).

Sowohl die Adressen werden berechnet als auch die Segmentierungs- und Paging-Mechanismen werden vom dem *Ausführung Modus* im Prozessor beeinflusst, wobei die x86-Verarbeitung direkt wichtige Ausführungsmodi unterstützt:

1. 16-Bit-*Real-Modus* (unter Verwendung eines 16-Bit-Speicherworts und einer begrenzten Sprache von 1 MB);
2. **16-Bit-geschützter 32-Bit-Modus**, gekennzeichnet durch die Verwendung von Paging und Segmentierung;
3. *8086 virtuellen Modus* ermöglicht die Anpassung von Programmen, um die realen Modi des berühmten geschützten Modus zu ändern;
4. Der *lange Modus* 64 sieht aus wie 32 Bit und ist schädlich, wenn die Segmentierung deaktiviert ist.

Während des Kurses konzentrieren wir uns auf die Architektur und das Verhalten von x86-Prozessoren im 32-Bit-geschützten Modus.

Die x86-Architektur ermöglicht die Verwendung von vier Segmenttypen mit

- *segment de cod*, care conține instrucțiuni mașină;
- *segment de date*, care conține date asupra cărora se acționează în conformitate cu instrucțiunile;
- *segment de stivă*;
- *segment suplimentar de date* (extrasegment).

Fiecare program este compus din unul sau mai multe segmente, de unul sau mai multe dintre tipurile de mai sus. În fiecare moment al execuției este activ cel mult câte un segment din fiecare tip. Regiștrii **CS** (*Code Segment*), **DS** (*Data Segment*), **SS** (*Stack Segment*), **ES** (*Extra Segment*) din **BIU** rețin valorile selectorilor segmentelor active, corespunzător fiecărui tip. Deci regiștrii CS, DS, SS și ES determină adresele de început și dimensiunile segmentelor active: de cod, de date, de stivă și suplimentar. Regiștrii **FS** și **GS** pot reține selectori indicând către segmente suplimentare, fără însă a avea roluri predeterminate. Datorită utilizării lor, CS, DS, SS, ES, FS și GS poartă denumirea de *regiștri de segment* (sau *regiștri selectori*). Registrul **EIP** (care oferă și posibilitatea accesării cuvântului său inferior prin subregistru **IP**) conține *offset-ul* instrucțiunii curente în cadrul segmentului de cod curent, el fiind manipulat exclusiv de către **BIU**.

Noțiunile asociate adresării sunt fundamentale pentru înțelegerea funcționării procesoarelor x86 și programării în limbaj de asamblare. Pentru aceasta, în Tabelul 1 le recapitulăm pe scurt în vederea clarificării:

unterschiedlichen Rollen:

- *Codesegment*, das MaschinenAnweisungen enthält;
- *Datensegment*, das Daten enthält, auf die es gemäß den Anweisungen einwirkt;
- *Stapelsegment*;
- *zusätzliches Datensegment* (Extrasegment).

Jedes Programm besteht aus einem oder mehreren Segmenten, einem oder mehreren der oben genannten Typen. Zu jedem Ausführungszeitpunkt ist höchstens ein Segment jedes Typs aktiv. Die Register **CS** (*Code Segment*), **DS** (*Data Segment*), **SS** (*Stack Segment*), **ES** (*Extra Segment*) in der **BIU** behalten die Werte der Selektoren der aktiven Segmenten bei, die jedem Typ entsprechen. Die Register CS, DS, SS und ES bestimmen also die Startadressen und die Dimensionen der aktiven Segmente: Code, Daten, Stapel und Additional. Die **FS**- und **GS**-Register können Selektoren behalten, die auf zusätzliche Segmente zeigen, ohne vorbestimmte Rollen zu haben. CS, DS, SS, ES, FS und GS werden aufgrund ihrer Verwendung als *Segmentregister* (oder *Selektorregister*) bezeichnet. Das **EIP**-Register (das auch die Möglichkeit bietet, über das **IP**-Unterregister auf sein unteres Wort zuzugreifen) enthält den *Offset* des aktuellen Anweisung innerhalb des aktuellen Codesegments, der ausschließlich von der **BIU** verarbeitet wird.

Die mit der Adressierung verbundenen Begriffe sind von grundlegender Bedeutung, um die Funktionsweise von x86-Prozessoren und die Programmierung in Assemblersprache zu verstehen. Zu diesem Zweck fassen wir sie in Tabelle 1 zusammen, um Folgendes zu verdeutlichen:

Tabelul 1. Adresarea în cadrul microprocesoarelor x8086 (Adressierung innerhalb von x8086-Mikroprozessoren)

Noțiune (Bedeutung)	Reprezentare (Darstellung)	Descriere (Beschreibung)
Specificare de adresă, adresă logică, adresă FAR (Adressangabe, logische Adresse, FAR-Adresse)	Selector ₁₆ :offset ₃₂ (Selektor ₁₆ :offset ₃₂)	Definește complet atât segmentul cât și deplasamentul în cadrul acestuia (Es definiert sowohl das Segment als auch die Verschiebung innerhalb des Segments vollständig)
Selector (Selektor)	16 biți	Identifică unul dintre segmentele disponibile. Ca valoare numerică acesta codifică poziția descriptorului de segment selectat în cadrul unei tabele de descriptori. (Identifiziert eines der verfügbaren Segmente. Als numerischer Wert codiert er die Position des ausgewählten Segmentdeskriptors in einer Deskriptortabelle.)
Offset, adresă NEAR (NEAR-Adresse)	Offset ₃₂	Definește doar componenta de offset (considerând segmentul cunoscut ori folosirea modelului de memorie flat) [Definiert nur die Offset-Komponente (unter Berücksichtigung des bekannten Segments oder unter Verwendung des Flat Memory-Modells)]
Adresă liniară (adresă de segmentare) [Lineare Adresse (Segmentierungadresse)]	32 biți	Început segment + offset, reprezintă rezultatul calculului de segmentare (Startsegment + Offset ist das Ergebnis der Segmentierungsberechnung)
Adresă fizică efectivă (Effektive physikalische Adresse)	Cel puțin 32 biți (Mindestens 32 Bit)	Rezultatul final al segmentării plus, eventual, paginării. Adresa finală obținută de către BIU, indicând în memoria fizică (hardware) [Das Endergebnis der Segmentierung plus möglicherweise die Paginierung. Die endgültige Adresse, die von der BIU erhalten wird und die im physischen Speicher (Hardware) angezeigt wird]

Diferențe cheie între paginare și segmentare

1. Diferența de bază între paginare și segmentare este că o pagină are întotdeauna **o dimensiune fixă a blocului**, în timp ce un segment este de **mărime variabilă**.
2. Pagingul poate duce la **fragmentarea internă** deoarece pagina este de dimensiune fixă a blocului, dar se poate întâmpla ca procesul să nu obțină întreaga dimensiune a blocului care va genera fragmentul intern din memorie. Segmentarea poate duce la **fragmentarea externă**, deoarece memoria este umplută cu blocurile cu dimensiuni variabile.
3. În paginare, utilizatorul oferă doar un **întreg întreg** ca adresă care este divizată de hardware într-un **număr de pagină și Offset**. Pe de altă parte, în segmentare

- utilizatorul specifică adresa în două cantități, adică **numărul segmentului și offsetul**.
4. Mărimea paginii este decisă sau specificată de **hardware**. Pe celelalte măini, dimensiunea segmentului este specificată de **utilizator**.
 5. În paginare, tabela de **pagini** hartă **adresa logică la adresa fizică** și conține adresa de bază a fiecărei pagini stocate în cadrele spațiului de memorie fizică. Cu toate acestea, în segmentare, **tabelul segmentului** hartă **adresa logică la adresa fizică** și conține numărul de segment și offset (limita segmentului).

5.2 Reprezentarea instrucțiunilor mașină

O instrucțiune mașină x86 reprezintă o secvență de 1 până la 15 octeți, care prin valorile lor specifică o operație de executat, operanții asupra cărora va fi aplicată, precum și modificatori suplimentari care controlează modul în care aceasta va fi executată.

O instrucțiune mașină x86 are maximum doi operanți. Pentru cele mai multe dintre instrucțiuni, cei doi operanți poartă numele de *sursă*, respectiv *destinație*. Dintre cei doi operanți, maximum unul se poate afla în memoria RAM. Celălalt se află fie într-un registru al **EU**, fie este o constantă întreagă. Astfel, o instrucțiune are forma:

Nume_instrucțiune destinație, sursă (Anweisung_name Ziel, Quelle)

Formatul intern al unei instrucțiuni este variabil, el putând ocupa între 1 și 15 octeți, având următoarea formă generală de reprezentare (*Instructions byte-codes from OllyDbg*):

[prefixe] + cod + [ModR/M] + [SIB] + [deplasament] + [imediat]
 ([Präfixe] + Code + [ModR / M] + [SIB] + [Offset] + [Sofort])

Prefixele controlează modul în care o instrucțiune se execută. Acestea sunt opționale (0 până la maximum 4) și ocupă câte un octet fiecare. De exemplu, acestea pot solicita execuția repetată (în buclă) a

5.2 Darstellung von Maschinenanweisungen

Ein x86-MaschinenAnweisung stellt eine Sequenz von 1 bis 15 Byte dar, die durch ihre Werte eine auszuführende Operation, die Operanden, auf die er angewendet wird, sowie zusätzliche Modifikatoren angibt, die steuern, wie er ausgeführt wird.

Ein x86-MaschinenAnweisung hat maximal zwei Operanden. Bei den meisten Anweisungen werden die beiden Operanden als *Quelle* bzw. *Ziel* bezeichnet. Maximal einer der beiden Operanden kann sich im RAM Speicher befinden. Die andere befindet sich entweder in einem EU-Register oder ist eine Ganzzahlkonstante. Eine Anweisung hat also die Form:

Das interne Format einer Anweisung ist variabel, es kann zwischen 1 und 15 Byte belegen und die folgende allgemeine Darstellungsform haben (*Instructions byte-codes from OllyDbg*):

Präfixe steuern, wie eine Anweisung ausgeführt wird. Diese sind optional (0 bis maximal 4) und belegen jeweils ein Byte. Beispielsweise können sie eine wiederholte Ausführung (Schleife) des aktuellen

instrucțiunii curente sau pot bloca magistrala de adrese pe parcursul execuției pentru a nu permite accesul concurrent la operanzi și rezultate.

Operația care se va efectua este identificată prin intermediul a 1 sau 2 octeți de *cod (opcode)*, aceștia fiind singurii octeți obligatoriu prezenți, indiferent de instrucțiune.

Anweisung erfordern oder den Adressbus während der Ausführung blockieren, um einen gleichzeitigen Zugriff auf Operanden und Ergebnisse zu verhindern.

Die auszuführende Operation wird durch 1 oder 2 Byte *Code (Opcode)* identifiziert, wobei dies die einzigen vorhandenen Bytes sind, unabhängig von der Anweisung.

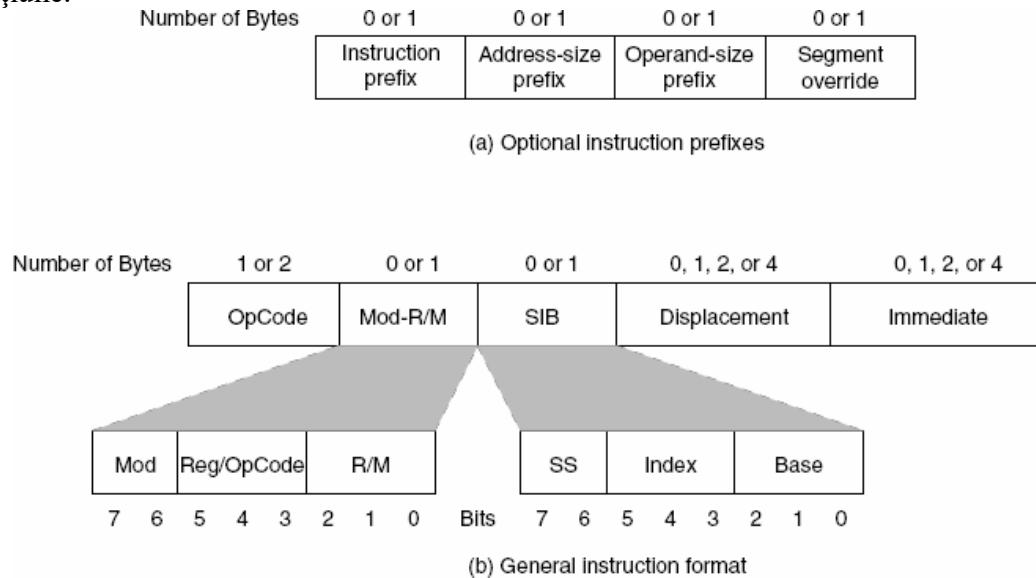


Figura 13. Prefixele și formatele instrucțiunilor microprocesorului 8086 (Präfixe und Formate für 8086-Mikroprozessoranweisungen)

Deși Figura 13 pare a indica faptul că instrucțiunile pot avea lungimi de până la 16 octeți, în realitate microprocesorul x86 nu permite instrucțiuni cu lungimi mai mari de 15 octeți.

Octetul *ModR/M* (mod registru/memorie) specifică pentru unele dintre instrucțiuni natura și locul operanzilor (registru, memorie, constantă întreagă etc.). Aceasta permite specificarea fie a unui registru, fie a unei locații de memorie a cărei adresă este exprimată prin intermediul unui *offset*.

Pentru cazuri mai complexe de adresare decât cele codificabile direct prin ModR/M, combinarea acestuia cu octetul SIB permite următoarea formulă generală de definire a unui *offset*:

Obwohl in Abbildung 13 anscheinend angegeben ist, dass die Anweisungen eine Länge von bis zu 16 Byte haben können, lässt der x86-Mikroprozessor in Wirklichkeit keine Anweisungen mit einer Länge von mehr als 15 Byte zu.

Das Byte ModR / M (Register- / Memory Modus) spezifiziert für einige Anweisungen die Art und den Ort der Operanden (Register, Speicher, Ganzzahlkonstante usw.). Es kann entweder ein Register oder ein Speicherort angegeben werden, dessen Adresse durch einen *Offset* ausgedrückt wird.

Bei komplexeren Adressierungsfällen als denen, die direkt von ModR/M codiert werden, ermöglicht die Kombination mit dem SIB-Byte die folgende allgemeine Formel zur Definition eines *Offsets*:

$$[\text{bază}] + [\text{index} \times \text{scală}] + [\text{constantă}] \\ ([\text{Basis}] + [\text{Index} \times \text{Skala}] + [\text{Konstante}])$$

unde pentru bază și index vor fi folosite valorile a doi registri iar scală este 1, 2, 4 sau 8. Regiștrii permisi ca bază sau / și index sunt: EAX, EBX, ECX, EDX, EBP, ESI, EDI. Registrul ESP este disponibil ca bază însă nu poate fi folosit cu rol de index.

Majoritatea instrucțiunilor folosesc pentru reprezentare fie numai câmpul de cod, fie cod urmat de ModR/M.

Deplasament (displacement) apare în cazul unor forme de adresare particulare (operanzi din memorie) și urmează direct după ModR/M sau SIB, când SIB este prezent. Acest câmp poate fi codificat fie pe octet fie pe dublucuvânt (32 biți).

Ca și consecință a imposibilității prezenței mai multor câmpuri de ModR/M, SIB și deplasament intr-o instrucțiune, arhitectura nu permite codificarea a două adrese de memorie în aceeași instrucțiune.

Valoare imediată oferă posibilitatea definirii unui operand ca fiind o constantă numerică pe 1, 2 sau 4 octeți. Când este prezent, acest câmp apare întotdeauna la sfârșitul instrucțiunii.

5.3 Adrese FAR și NEAR

Pentru a adresa o locație din memoria RAM sunt necesare două valori: una care să indice segmentul, alta care să indice *offset-ul* în cadrul segmentului. Pentru a simplifica referirea la memorie, microprocesorul derivă, în lipsa unei alte specificări, adresa segmentului din unul dintre regiștrii de segment CS, DS, SS sau ES. Alegerea implicită a unui registru de segment se face după niște reguli proprii instrucțiunii folosite.

Wobei für die Basis und den Index die Werte von zwei Registern verwendet werden und die Skala 1, 2, 4 oder 8 ist. Die zulässigen Register als Basis und / oder Index sind: EAX, EBX, ECX, EDX, EBP, ESI, EDI. Das ESP-Register ist als Basis verfügbar, kann jedoch nicht als Index verwendet werden.

Die meisten Anweisungen verwenden zur Darstellung entweder nur das Codefeld oder den Code, gefolgt von ModR / M.

Verschiebung (displacement) tritt bei bestimmten Adressierungsformen (Operanden aus dem Speicher) auf und folgt unmittelbar nach ModR/M oder SIB, wenn SIB vorhanden ist. Dieses Feld kann entweder byteweise oder durch Doppelwort (32 Bit) codiert werden.

Infolge der Unmöglichkeit, dass mehrere Felder von ModR/M, SIB und Verschiebung in einem Befehl vorhanden sind, erlaubt die Architektur nicht die Codierung von zwei Speicheradressen in dieselbe Anweisung.

Der *Sofortwert* bietet die Möglichkeit, einen Operanden als numerische Konstante auf 1, 2 oder 4 Bytes zu definieren. Wenn vorhanden, erscheint dieses Feld immer am Ende der Anweisung.

5.3 FAR- und NEAR-Adressen

Um eine Position im RAM zu adressieren, sind zwei Werte erforderlich: einer zur Angabe des Segments und ein anderer zur Angabe des *Offsets* innerhalb des Segments. Um die Speicherreferenz zu vereinfachen, leitet der Mikroprozessor, sofern nicht anders angegeben, die Segmentadresse aus einem der Segmentregister CS, DS, SS oder ES ab. Die Standardauswahl eines Segmentregisters erfolgt nach bestimmten Regeln, die für die verwendete Anweisung spezifisch sind.

Prin definiție, o adresă în care se specifică doar *offset*-ul, urmând ca segmentul să fie preluat implicit dintr-un registru de segment poartă numele de *adresă NEAR* (adresă apropiată). O adresă NEAR se află întotdeauna în interiorul unuia din cele patru segmente active.

O adresă în care programatorul indică explicit un selector de segment poartă numele de *adresă FAR* (adresă îndepărtată). O adresă FAR este deci o SPECIFICARE COMPLETĂ DE ADRESĂ și ea se poate exprima în trei moduri:

- $s_3s_2s_1s_0$: specificare_offset unde $s_3s_2s_1s_0$ este o constantă;
- registru_segment:specificare_offset, registru segment fiind CS, DS, SS, ES, FS sau GS;
- FAR [variabilă], unde variabilă este de tip QWORD și conține cei 6 octeți constituind adresa FAR.

Formatul intern al unei adrese FAR este: la adresa mai mică se află offsetul, iar la adresa mai mare cu 4 (cuvântul care urmează după dublucuvântul curent) se află cuvântul care conține selectorul care indică segmentul.

Reprezentarea adreselor respectă principiul reprezentării little-endian: partea cea mai puțin semnificativă are adresa cea mai mică, iar partea cea mai semnificativă are adresa cea mai mare.

5.4 Calculul offset-ului unui operand. Moduri de adresare

În cadrul unei instrucțiuni există 3 moduri de a specifica un operand pe care aceasta îl solicită:

- *modul registru*, dacă pe post de operand se află un registru al mașinii;
- *modul imediat*, atunci când în instrucțiune se află chiar valoarea operandului (nu adresa lui și nici un registru în care să fie conținut);

Per Definition trägt eine Adresse, bei der nur der *Offset* angegeben ist, gefolgt von dem Segment, das standardmäßig aus einem Segmentregister entnommen wird, den *NEAR-Adressennamen*. Eine NEAR-Adresse befindet sich immer in einem der vier aktiven Segmente.

Eine Adresse, bei der der Programmierer ausdrücklich einen Segmentwähler angibt, wird als *FAR-Adresse* (ferne Adresse) bezeichnet. Eine FAR-Adresse ist daher eine vollständige Adressangabe und kann auf drei Arten ausgedrückt werden:

- $s_3s_2s_1s_0$: Offset-Angabe, wobei $s_3s_2s_1s_0$ eine Konstante ist;
- Segmentregister:Offset-Spezifikation, wobei das Segmentregister CS, DS, SS, ES, FS oder GS ist;
- FAR [Variable], wobei die Variable vom Typ QWORD ist und die 6 Bytes enthält, die die FAR-Adresse bilden.

Das interne Format einer FAR-Adresse lautet: Bei der niedrigeren Adresse ist der Offset und bei der größeren Adresse mit 4 (dem Wort nach dem aktuellen Doppelwort) das Wort, das den Selektor enthält, der das Segment angibt.

Die Adressendarstellung respektiert das Prinzip der Little-Endian-Darstellung: Der niedrigstwertige Teil hat die kleinste Adresse und der höchstwertige Teil hat die höchste Adresse.

5.4 Berechnung des *Offsets* eines Operanden. Adressierungsarten

Innerhalb einer Anweisung gibt es drei Möglichkeiten, einen angeforderten Operanden anzugeben:

- der *Registermodus*, wenn sich ein Maschinenregister auf dem Operanden befindet;
- der *unmittelbare Modus*, wenn sich der Wert des Operanden in die Anweisung befindet (nicht seine Adresse und kein

- *modul adresare la memorie*, dacă operandul se află efectiv undeva în memorie. În acest caz, adresa *offset*-ului lui se calculează după următoarea formulă:

$$\text{adresa_offset} = [\text{bază}] + [\text{index} \times \text{scală}] + [\text{constanta}] \\ (\text{Offset_Adresse} = [\text{Basis}] + [\text{Index} \times \text{Skala}] + [\text{Konstante}])$$

Deci *adresa_offset* se obține din următoarele (maxim) patru elemente:

- conținutul unuia dintre registrii EAX, EBX, ECX, EDX, EBP, ESI, EDI sau ESP ca bază;
- conținutul unuia dintre registrii EAX, EBX, ECX, EDX, EBP, ESI sau EDI drept index;
- factor numeric (scală) pentru a înmulți valoarea registrului index cu 1, 2, 4 sau 8
- valoarea unei constante numerice, pe octet sau dublucuvânt.

De aici rezultă următoarele moduri de adresare a memoriei:

- *directă*, atunci când apare numai *constanta*;
- *bazată*, dacă în calcul apare unul dintre registrii *bază*;
- *scalat-indexată*, dacă în calcul apare unul dintre registrii *index*;

Cele trei moduri de adresare a memoriei pot fi combinate. De exemplu, poate să apară adresare directă bazată, adresare bazată și scalat-indexată etc.

La instrucțiunile de salt mai apare și un alt tip de adresare numit adresare *relativă*.

Adresa relativă indică poziția următoarei instrucțiuni de executat, în raport cu poziția curentă. Poziția este indicată prin numărul de octeți de cod peste care se va sări. Arhitectura x86 permite atât adrese relative

- Register, in dem er enthalten ist);
- *Speicheradressierungsmodus*, wenn sich der Operand tatsächlich irgendwo im Speicher befindet. In diesem Fall wird die Adresse seines *Offsets* nach folgender Formel berechnet:

$$\text{adresa_offset} = [\text{bază}] + [\text{index} \times \text{scală}] + [\text{constanta}] \\ (\text{Offset_Adresse} = [\text{Basis}] + [\text{Index} \times \text{Skala}] + [\text{Konstante}])$$

Daher wird die *Offset_Adresse* aus den folgenden (maximal) vier Elementen erhalten:

- der Inhalt eines der EAX-, EBX-, ECX-, EDX-, EBP-, ESI-, EDI- oder ESP-Register als Basis;
- der Inhalt eines der Register EAX, EBX, ECX, EDX, EBP, ESI oder EDI als Index;
- numerischer Faktor (Skala), um den Wert des Indexregisters mit 1, 2, 4 oder 8 zu multiplizieren;
- der Wert einer numerischen Konstante pro Byte oder Doppelwort.

Ab hier ergeben sich folgende Möglichkeiten zur Adressierung des Speichers:

- *direkt*, wenn nur die *Konstante* erscheint;
- *basierend*, wenn eines der Basisregister in der Berechnung erscheint;
- *skaliert-indiziert*, wenn eines der Indexregister in der Berechnung erscheint;

Die drei Adressierungsarten des Speichers können kombiniert werden. Beispielsweise können eine direktbasierte Adressierung, eine basierte Adressierung und eine skalierte- *indizierte* Adressierung usw.

Auftreten eine andere Art von Adresse, die als *relative* Adressierung bezeichnet wird, erscheint auch in den SprungAnweisungen. Die *relative Adresse* gibt die Position der folgenden auszuführenden Anweisung relativ zur aktuellen Position an. Die Position wird durch die Anzahl der zu überspringenden Code-Bytes angegeben. Die x86-Architektur ermöglicht sowohl

scurte (*SHORT Addresses*), reprezentate pe octet și având valori între -128 și 127, cât și adrese relative apropriate (*NEAR Addresses*), pe dublucuvânt cu valori între -2147483648 și 2147483647.

kurze relative Adressen (*SHORT Addresses*), die durch Byte dargestellt werden und Werte zwischen -128 und 127 aufweisen, als auch nahe relative Adressen (*NEAR Addresses*) in einem Duplikat mit Werten zwischen -2147483648 und 2147483647.

Curs 5 și 6

Contents

1. Elementele de bază ale limbajului de asamblare	3
1. Die Grundelemente der Assemblersprache	3
1.1 Limbaj simbolic. Simboluri. Mnemonice și etichete	3
1.1 Symbolsprache. Symbole. Mnemonik und Beschriftungen.....	3
1.2 Formatul unei linii sursă.....	4
1.2 Das Format einer Quellzeile.....	4
2. Expresii.....	6
2. Ausdrücke.....	6
2.1 Moduri de adresare	7
2.1 Adressierungsarten	7
2.1.1 Utilizarea operanzilor imediați	7
2.1.1 Verwendung von Sofortoperanden	7
2.1.2 Utilizarea operanzilor regisztru	10
2.1.2 Verwendung von Registrierungsoperanden.....	10
2.1.3 Utilizarea operanzilor din memorie	10
2.1.3 Verwenden der Operanden aus dem Speicher	10
2.2 Utilizarea operatorilor	14
2.2 Verwendung von Operatoren	14
2.2.1 Operatori de deplasare de biți	15
2.2.1 Bitweise Verschiebungoperatoren	15
2.2.2 Operatori logici pe biți	15
2.2.2 Bitweise Logischeoperatoren.....	15
2.2.3 Operatorul de specificare a segmentului	16
2.2.3 Der Segmentspezifikationsoperator	16
2.2.4 Operatori de tip	16
2.2.4 Typoperatoren	16
3. Directive	18
3. Anweisungen	18
3.1 Directiva SEGMENT.....	18
3.1 Die SEGMENT-Anweisung.....	18

3.2 Directiva ASSUME	20
3.2 Die ASSUME-Anweisung	20
3.3 Directive pentru definirea datelor.....	22
3.3 Anweisungen zur Datendefinition.....	22
3.4 Directiva EQU.....	25
3.4 Die EQU-Anweisung.....	25
3.5 Directivele LABEL și PROC	26
3.5 LABEL- und PROC- Anweisungen	26
3.6 Blocuri repetitive	28
3.6 Wiederholungsblöcke	28
3.7 Directiva INCLUDE	30
3.7 Die INCLUDE-Anweisung	30
3.8 Macrouri.....	31
3.8 Makros	31

1. Elementele de bază ale limbajului de asamblare

Limbajul de asamblare al unui calculator este un limbaj de programare în care setul de bază al instrucțiunilor coincide cu operațiile mașinii și ale cărui structuri de date coincid cu structurile primare de date ale mașinii.

Limbajul mașină al unui sistem de calcul (SC) este format din totalitatea instrucțiunilor mașină puse la dispoziție de procesorul SC. Acestea se reprezintă sub forma unor siruri de biți cu semnificație prestabilită.

1.1 Limbaj simbolic. Simboluri. Mnemonice și etichete

Elementele cu care lucrează un asamblor sunt:

- etichete – nume scrise de utilizator, cu ajutorul căror se pot referi date sau zone de memorie.
- instrucțiuni – scrise sub forma unor mnemonice care sugerează acțiunea. Asamblorul generează octeții care codifică instrucțiunea respectivă.
- directive – sunt indicații date asamblorului în scopul generării corecte a octeților. Exemplu: relații între modulele obiect, definirea unor segmente, indicații de asamblare condiționată, directive de generare a datelor.
- contor de locații (*program counter*) – număr întreg gestionat de asamblor. În fiecare moment, valoarea contorului coincide cu numărul de octeți generați corespunzător instrucțiunilor și directivelor deja întâlnite în cadrul segmentului respectiv (deplasamentul curent în cadrul segmentului).

1. Die Grundelemente der Assemblersprache

Die Assemblersprache eines Computers ist eine Programmiersprache, in der der grundlegende Anweisungssatz mit den Operationen der Maschine übereinstimmt und deren Datenstrukturen mit den primären Datenstrukturen der Maschine übereinstimmen.

Die Maschinensprache eines Computersystems (CS) besteht aus allen vom CS-Prozessor bereitgestellten MaschinenAnweisungen. Diese werden in Form von Bitfolgen mit vorgegebener Signifikanz dargestellt.

1.1 Symbolsprache. Symbole. Mnemonik und Beschriftungen

Die Elemente, mit denen eine Assembler arbeitet, sind:

- Label – vom Benutzer geschriebene Namen, mit denen auf Daten oder Speicherbereiche verwiesen werden kann.
- Anweisungen – in Form von Mnemonik geschrieben, die die Aktion vorschlagen. Der Assembler generiert die Bytes, die den jeweiligen Anweisung codieren.
- Direktiven – sind Angaben zur Assembler, um die korrekten Bytes zu generieren. Beispiel: Beziehungen zwischen Objektmodulen, Definition von Segmenten, bedingte Assemblierungsanweisungen, Anweisungen zur Datengenerierung.
- Positionsähler, Programmzähler oder Befehlszähler (*program counter*) – ganze Zahl, die von der Assembler verwaltet wird. Zu jedem Zeitpunkt stimmt der Wert des Programmzählers mit der Anzahl der Bytes überein, die gemäß den Anweisungen und Anweisungen generiert wurden, die bereits in dem

Programatorul poate utiliza această valoare (accesare doar în citire!) prin simbolul '\$'.

Asamblorul suportă două simboluri speciale („tokens”) în expresii, cu ajutorul cărora se pot efectua calcule care implică poziția curentă la care a ajuns execuția programului scris în limbaj de asamblare: este vorba despre „\$” și „\$\$. Token-ul „\$” reprezintă poziția curentă a liniei care conține expresia cu „\$”. Deci putem codifica o buclă infinită dacă scriem „JMP \$”.

Token-ul „\$\$” reprezintă începutul secțiunii curente. Așadar, putem determina cât de departe am ajuns în interiorul secțiunii folosind expresia „(\$-\$”)”.

1.2 Formatul unei linii sursă

Formatul unei linii sursă în limbajul de asamblare x86 este următorul:

[etichetă[:]] [prefixe] [mnemonică] [operanzi] [;comentariu]
([Label [:]] [Präfixe] [Mnemonik] [Operanden] [; Kommentar])

Ilustrăm conceptul prin intermediul a câteva exemple de linii sursă:

<i>hier: jmp hier</i>	; haben wir Label + Mnemonik + operand + Kommentar
repz cmpsd	; Präfix + Mnemonik + Kommentar
<i>start:</i>	; Label + Kommentar
<i>a dw 19872, 42h</i>	; nur ein Kommentar (care putea lipsi și el - was fehlen könnte)
	; Label + Mnemonik + 2 Operanden + Kommentar

Caracterele din care poate fi constituită o etichetă sunt următoarele:

Litere, atât A-Z cât și a-z;
Cifre de la 0 la 9;
Caracterele _, \$, #, @, ~, . și ?

Ca prim caracter al unei etichete sunt

jeweiligen Segment angetroffen wurden (die aktuelle Offset innerhalb des Segments). Der Programmierer kann diesen Wert (Nur-Lese-Zugriff!) mit dem Symbol '\$' verwenden.

Der Assembler unterstützt zwei spezielle Symbole („Tokens“) in den Ausdrücken, mit denen Berechnungen durchgeführt werden können, die die aktuelle Position implizieren, an der die Ausführung des geschriebenen Programms in Assemblersprache angekommen ist: „\$“ und „\$\$“. Das Token „\$“ repräsentiert die aktuelle Position der Zeile, die den Ausdruck „\$“ enthält. Wir können also eine Endlosschleife codieren, wenn wir „JMP \$“ schreiben.

Das Token „\$\$“ ist der Anfang des aktuellen Abschnitts. Anhand des Ausdrucks „(\$ - \$\$“ können wir also feststellen, wie weit wir innerhalb des Abschnitts sind.

1.2 Das Format einer Quellzeile

Das Format einer Quellzeile in der x86-Assemblersprache lautet wie folgt:

Wir veranschaulichen das Konzept anhand einiger Beispiele für Quelltextzeilen:

Die Zeichen, aus denen ein Label gebildet werden kann, sind die folgenden:

Buchstaben von A bis Z und von a bis z;
Zahlen von 0 bis 9;
Die Zeichen _, \$, #, @, ~, . und ?

Als erstes Zeichen eines Labels sind nur die

permise doar litere, _ și ?

Aceste reguli sunt valabile pentru toți *identificatorii* valizi (denumiri simbolice, precum nume de variabile, etichete, macro, etc.).

Identifierii definiți de utilizator sunt *case sensitive*: limbajul face distincție între literele mari și cele mici. Aceasta înseamnă că un identificator „Abc” este diferit de identificatorul „abc”. Pentru denumirile care fac implicit parte din limbaj, cum ar fi cuvintele cheie, mnemonicile și numele reștrictorilor, nu se diferențiază literele mari de cele mici (acestea sunt *case insensitive*). La nivelul limbajului de asamblare se întâlnesc două categorii de etichete:

1) **etichete de cod**, care apar în cadrul secvențelor de instrucțiuni (segmente de cod) cu scopul de a defini destinațiile de transfer ale controlului în cadrul unui program.

2) **etichete de date**, care identifică simbolic unele locații de memorie, din punct de vedere semantic ele fiind echivalentul noțiunii de *variabilă* din alte limbaje.

Valoarea unei etichete în limbaj de asamblare este un număr întreg reprezentând adresa instrucțiunii, directivei sau datelor care urmează etichetei.

Distincția dintre referirea adresei unei variabile sau a conținutului asociat acesteia se face după regulile:

Când este specificat între paranteze drepte, numele variabilei desemnează valoarea variabilei, de exemplu [p] specifică accesarea valorii variabilei p, similar cu modul în care *p semnifică dereferențierea unui pointer (accesul la conținutul indicat prin valoarea pointerului) în C;

În orice alt context numele variabilei reprezintă adresa variabilei, spre exemplu, p este întotdeauna adresa variabilei p;

Buchstaben _ und? zulässig.

Diese Regeln gelten für alle gültigen *Bezeichner* (symbolische Namen wie Variablennamen, Bezeichnungen, Makros usw.).

Benutzerdefinierte Bezeichner sind *case sensitive*: die Assemblersprache unterscheidt zwischen Groß- und Kleinschreibung. Dies bedeutet, dass sich eine „Abc“ Bezeichner von der „abc“ Bezeichner unterscheidet. Bei den Namen, die implizit Teil der Sprache sind, wie Schlüsselwörter, Mnemonik und Registrarnamen, unterscheiden sich Groß- und Kleinbuchstaben nicht (dies sind *case insensitive*).

Auf Assemblersprachenebene gibt es zwei Kategorien von Labels:

1) **Code-Labels**, die in den Anweisungssequenzen (Codesegmenten) erscheinen, um die Steuerübertragungsziele innerhalb eines Programms zu definieren.

2) **Daten-Labels**, die einige Speicherorte symbolisch kennzeichnen, semantisch dem Begriff der *Variablen* in anderen Sprachen entsprechen.

Der Wert eines Labels in Assemblersprache ist eine Ganzzahl, die die Adresse der Anweisung, Direktive oder Daten darstellt, die auf das Label folgen.

Die Unterscheidung zwischen dem Verweis auf die Adresse einer Variablen oder ihrem Inhalt erfolgt nach folgenden Regeln:

Wenn in eckigen Klammern angegeben, gibt der Variablename den Wert der Variablen an. Beispielsweise gibt [p] den Zugriff auf den Wert der Variablen p an, ähnlich wie *p eine Zeiger-Dereferenzierung (Zugriff auf den durch den Zeigerwert angegebenen Inhalt) in C angibt.

In jedem anderen Kontext repräsentiert der Name der Variablen die Adresse der Variablen, zum Beispiel ist p immer die

Exemple:

mov EAX, et ; încarcă în registrul EAX
adresa datelor sau a codului
marcat cu eticheta „et”

mov EAX, [et]; încarcă în registrul EAX
conținutul de la adresa „et”
(4 octeți)

Adresse der Variablen *p*;

Beispiele:

mov EAX, et ; ladet in das EAX-Register
die Daten- oder den
Code-Adresse, der mit dem
Label „et“ markiert ist

mov EAX, [et] ; ladet den Inhalt von der
Adresse „et“ (4 Bytes) in das
EAX-Register

Folosirea parantezelor pătrate indică întotdeauna accesarea unui operand din memorie.

De exemplu, **mov EAX, [EBX]** semnifică un transfer în EAX a conținutului memoriei a cărei adresă este datează de valoarea lui EBX.

Există două tipuri de *mnemonice*: mnemonice de *instrucțiuni* și nume de *directive*. Directivele dirijează asamblorul. Ele specifică modul în care asamblorul va genera codul obiect. Instrucțiunile dirijează procesorul.

Operanții sunt parametri care definesc valorile ce vor fi prelucrate de instrucțiuni sau de directive. Ei pot fi **registri**, **constante**, **etichete**, **expresii**, **cuvinte cheie sau alte simboluri**. Semnificația operanților depinde de mnemonica instrucțiunii sau directivei asociate.

2. Expresii

O expresie este formată din operanzi și operatori. Operatorii indică modul de combinare a operanților în scopul formării expresiei. Expresiile sunt evaluate în momentul asamblării (adică, valorile lor sunt determinabile la momentul asamblării, cu excepția acelor părți care desemnează conținuturi de registri și care vor fi determinate la execuție).

Die Verwendung von eckigen Klammern zeigt immer an, dass auf einen Operanden aus dem Speicher zugegriffen wird.

Zum Beispiel bezeichnet **mov EAX, [EBX]** eine Übertragung des Speicherinhalts, dessen Adresse durch den Wert von EBX gegeben ist, an EAX.

Es gibt zwei Arten von *Mnemoniken*: Mnemoniken von Anweisungen und Namen von Anweisungen. Die Direktiven leiten den Assembler. Sie geben an, wie der Assembler den Objektcode generiert. Die Anweisungen leiten den Prozessor.

Operanden sind Parameter, die die Werte definieren, die von Anweisungen oder Direktiven verarbeitet werden. Dies können **Register**, **Konstanten**, **Bezeichnungen**, **Ausdrücke**, **Schlüsselwörter** oder **andere Symbole** sein. Die Bedeutung der Operanden hängt von der Mnemonik der zugehörigen Anweisung oder Direktive ab.

2. Ausdrücke

Ein Ausdruck besteht aus Operanden und Operatoren. Die Operatoren geben an, wie die Operanden zum Ausdruck zusammengefasst werden. Die Ausdrücke werden zum Zeitpunkt des Assemblierung ausgewertet (dh ihre Werte sind zum Zeitpunkt des Assemblierung bestimmbar, mit Ausnahme derjenigen Teile, die den Inhalt von Registern bezeichnen und die bei

der Ausführung bestimmt werden).

2.1 Moduri de adresare

Operanzii instrucțiunilor pot fi specificați sub forme numite **moduri de adresare**.

Cele trei tipuri de operanzi sunt: **operaṇzi imediați**, **operaṇzi registru** și **operaṇzi în memorie**.

Valoarea operanzilor este calculată în momentul asamblării pentru operanzii imediați, în momentul încărcării programului pentru adresarea directă (adresa FAR) și în momentul execuției pentru operanzii registru și cei adresați indirect.

2.1.1 Utilizarea operaṇzilor imediați

Operanzii imediați sunt formați din date numerice constante calculabile la momentul asamblării.

Constantele sunt utilizate ca operaṇzi în expresii. Limbajul de asamblare recunoaște patru tipuri de valori constante: **întręgi**, **șiruri de caractere**, **numere reale** și **constante împachetate codificate binar zecimal (BCD)**.

Constantele întregi se specifică prin valori binare, octale, zecimale sau hexazecimale. Adițional, este permisă folosirea caracterului _ (*underscore*) pentru a separa grupuri de cifre. Baza de numerație poate fi precizată în mai multe moduri:

- Folosind sufixe H sau X pentru hexazecimal, D sau T pentru zecimal, Q sau O pentru octal și B sau Y pentru binar. În aceste cazuri, numărul trebuie neapărat să înceapă cu o cifră între 0 și 9, pentru a nu exista confuzii între

2.1 Adressierungsarten

Befehlsoperanden können in Form von **Adressierungsmodi** angegeben werden.

Die drei Arten von Operanden sind: **Sofortoperanden**, **Registeroperanden** und **Speicheroperanden**.

Der Wert der Operanden wird zum Zeitpunkt des Assemblierungs für die unmittelbaren Operanden, zum Zeitpunkt des Ladens des Programms für die direkte Adressierung (FAR-Adresse) und zum Zeitpunkt der Ausführung für das Operandenregister und die indirekt adressierten Operanden berechnet.

2.1.1 Verwendung von Sofortoperanden

Sofortige Operanden bestehen aus dem konstanten numerischen Daten, das zum Zeitpunkt der Assemblierung berechnet werden kann.

Konstanten sind als Operanden in Ausdrücken verwendet. In der Assemblersprache werden vier Arten von Konstantenwerten erkannt: **Ganzzahlen**, **Zeichenfolgen**, **reelle Zahlen** und **binär kodierte Dezimalzahlen (BCD)**.

Ganze Konstanten sind spezifisch für Binär-, Oktal-, Dezimal- oder Hexadezimalwerte. Zusätzlich ist der Unterstrich von der Zeichen _ (*underscore*) für eine separate Zifferngruppe zulässig. Die Zahlenbasis kann auf verschiedene Arten angegeben werden:

- Verwenden die Suffixen H oder X für hexadezimal, D oder T für dezimal, Q oder O für oktal und B oder Y für binär. In diesen Fällen muss die Zahl unbedingt mit einer Zahl zwischen 0 und 9 beginnen, damit Konstanten und

constante și simboluri. De exemplu, 0ABCH este interpretat ca număr hexazecimal, dar ABCH este interpretat ca simbol.

- În stilul specific limbajului C, prin prefixare cu 0x sau 0h pentru hexazecimal, 0d sau 0t pentru zecimal, 0o sau 0q pentru octal, respectiv 0b sau 0y pentru binar;

Exemple:

- constanta hexazecimală B2A poate fi exprimată ca 0xb2a, 0xb2A, 0hb2a, 0b12Ah, 0B12AH, etc;
- valoarea zecimală 123 poate fi specificată ca 123, 0d123, 0d0123, 123d, 123D, ...
- 11001000b, 0b11001000, 0y1100_1000, 001100_1000Y reprezintă diferite exprimări ale numărului binar 11001000.

O constantă de tip sir este formată din unul sau mai multe caractere ASCII delimitate de ghilimele sau de apostrofuri. Dacă printre aceste caractere ASCII trebuie să apară caracterul delimitator, acesta trebuie dublat. Exemple: 'a', "a", "Ia vino' 'ncoa".

Pentru unele instrucțiuni există o limită maximă în ceea ce privește dimensiunea de reprezentare a valorilor imediate (de obicei 8, 16 sau 32 de biți). Constantele de tip sir care sunt mai lungi de două caractere (patru caractere la procesoarele 80386) nu pot fi date imediate. Ele trebuie să fie stocate în memorie înainte de a fi prelucrate de instrucțiuni.

Datele imediate nu sunt admise ca operand destinație (așa cum nici 2:=N nu este o instrucțiune validă în nici un limbaj de programare de nivel înalt).

Constantele întregi codificate zecimal

Symbole nicht verwechselt werden. Beispielsweise wird 0ABCH als Hexadezimalzahl interpretiert, ABCH jedoch als Symbol.

- In dem für die Sprache C spezifischen Stil mit dem Präfix 0x oder 0h für hexadezimal, 0d oder 0t für dezimal, 0o oder 0q für oktal bzw. 0b oder 0y für binär;

Beispiele:

- Die hexadezimale Konstante B2A kann ausgedrückt werden als 0xb2a, 0xb2A, 0hb2a, 0b12Ah, 0B12AH usw.;
- Der Dezimalwert 123 kann als 123, 0d123, 0d0123, 123d, 123D, ... angegeben werden;
- 11001000b, 0b11001000, 0y1100_1000, 001100_1000Y stellen unterschiedliche Ausdrücke der Binärzahl 11001000 dar.

Eine Zeichenfolgekonstante besteht aus einem oder mehreren ASCII-Zeichen, die durch Anführungszeichen oder Apostrophe getrennt sind. Wenn das Trennzeichen zwischen diesen ASCII-Zeichen stehen muss, muss es verdoppelt werden. Beispiele: 'a', "a", "Ia vino' 'ncoa".

Für einige Anweisungen gibt es eine maximale Grenze für die Größe der Sofortwertdarstellung (normalerweise 8, 16 oder 32 Bit). Zeichenfolgentypkonstanten, die länger als zwei Zeichen sind (vier Zeichen bei 80386-Prozessoren), können nicht sofortige Daten werden. Sie müssen gespeichert werden, bevor sie von den Anweisungen verarbeitet werden.

Sofortige Daten sind als Zieloperand nicht zulässig (da entweder 2:=N keine gültige Anweisung in einer höheren Programmiersprache ist).

Dezimal codierte Ganzahlkonstanten sind

constituie un tip special de constante care pot fi utilizate numai pentru inițializarea variabilelor BCD.

Cu *numere reale* se poate opera numai în prezența unui coprocesor matematic.

Deplasamentele etichetelor de date și de cod reprezintă valori determinabile la momentul asamblării care rămân **constante** pe tot parcursul execuției programului.

De exemplu, o instrucțiune de genul:

mov EAX, et; transfer în registrul EAX a adresei efective asociate etichetei *et*

va putea fi evaluată la momentul asamblării drept, de exemplu,

mov eax, 8 ; distanță de 8 octeți față de începutul segmentului de date

„Constanța” acestor valori derivă din regulile de alocare adoptate de limbajele de programare în general și care statuează că ordinea de alocare în memorie a variabilelor declarate (mai precis distanța față de începutul segmentului de date în care o variabilă este alocată) sau respectiv distanțele salturilor destinație în cazul unor instrucțiuni de tip **goto** sunt valori constante pe parcursul execuției unui program.

Adică: o variabilă odată alocată în cadrul unui segment de memorie nu își va schimba niciodată locul alocării (adică poziția sa față de începutul aceluia segment). Această informație poate fi determinată la momentul asamblării; ea derivă din ordinea specificării variabilelor la declarare în cadrul textului sursă și din dimensiunea de reprezentare (dedusă pe bază informației de tip asociate).

eine spezielle Art von Konstanten, die nur zum Initialisieren von BCD-Variablen verwendet werden können.

Mit *reellen Zahlen* können Sie nur in Gegenwart eines mathematischen Coprozessors arbeiten.

Die Offseten der Daten- und Code-Label stellen bestimmbare Werte zum Zeitpunkt der Zusammenstellung dar, die während der Ausführung des Programms **konstant bleiben**.

Zum Beispiel eine Anweisung wie:

mov EAX, et; Übertragen der dem Label *et* zugeordneten tatsächlichen Adresse in das EAX-Register

kann zum Zeitpunkt der Assemblierung ausgewertet werden als, zu Beispiel:

mov eax, 8; 8 Byte Abstand vom Beginn des Datensegments

Die „Konstanz“ dieser Werte ergibt sich aus den von den Programmiersprachen allgemein übernommenen Zuordnungsregeln und der Reihenfolge der Zuordnung der deklarierten Variablen (genauer der Abstand vom Beginn des Datensegments, in dem eine Variable zugeordnet ist) bzw. den Sprungdistanzen im Speicher Ziel bei **goto**-Anweisungen sind konstante Werte während der Ausführung eines Programms.

Das heißt: Eine Variable, die einmal innerhalb eines Speichersegments zugewiesen wurde, ändert niemals ihren Zuweisungsort (deshalb ihre Position relativ zum Beginn dieses Segments). Diese Informationen können zum Zeitpunkt der Assemblierung ermittelt werden; sie ergibt sich aus der Reihenfolge der Angabe der zu deklarierenden Variablen im Quelltext und aus der Dimension der Darstellung (abgeleitet aus den zugehörigen Typinformationen).

2.1.2 Utilizarea operanzilor registru

Modul de *adresare directă*:

mov EAX, EBX; provoacă transferul valorii din registrul BX în registrul AX.

Adresare indirectă, pentru a indica locațiile de memorie:

mov EAX, [EBX]; provoacă transferul cuvântului de memorie de la adresa desemnată de registrul BX spre registrul AX.

2.1.3 Utilizarea operanzilor din memorie

Operanzii din memorie sunt cu *adresare directă* și cu *adresare indirectă*.

Operandul cu *adresare directă* este o constantă sau un simbol care reprezintă adresa (segment și deplasament) unei instrucțiuni sau a unor date. Acești operanzi pot fi *etichete* (de ex: **jmp** et), *nume de proceduri* (de ex: **call proc1**) sau *valoarea contorului de locații* (de ex: b db \$-a).

Deplasamentul unui operand cu adresare directă este calculat în momentul asamblării (adică la „assembly time”). Adresa fiecărui operand raportată la structura programului executabil (mai precis stabilirea segmentelor la care se raportează deplasamentele calculate) este calculată **în momentul editării de legături** (*linking time*). Adresa fizică efectivă este calculată **în momentul încărcării programului pentru execuție** (*loading time*).

Adresa efectivă este întotdeauna raportată la un registru de segment. Acest registru poate fi specificat explicit sau, în caz

2.1.2 Verwendung von Registrierungsoperanden

Direktadressierungsmodus:

mov EAX, EBX; bewirkt die Übertragung des Wertes vom EBX-Register in das EAX-Register.

Indirekte Adressierung zur Angabe von Speicherplätzen:

mov EAX, [EBX]; Bewirkt die Wortübertragung von der vom EBX-Register angegebenen Adresse zum EAX-Register.

2.1.3 Verwenden der Operanden aus dem Speicher

Die Operanden im Speicher haben eine *direkte* und eine *indirekte Adressierung*.

Der *direkte Adressierungsoperand* ist eine Konstante oder ein Symbol, das die Adresse (Segment und Verschiebung) eines Anweisungs oder von Daten darstellt. Diese Operanden können *Labels* (zB: **jmp** et), *Prozedurnamen* (zB: **call proc1**) oder *der Wert des Programmzählers* (zB: b db \$-a) sein.

Das Offset eines direkten Adressoperanden wird zum Zeitpunkt des Assemblierungs (dh zum „assembly time“) **berechnet**. Die Adresse jedes Operanden, die sich auf die Struktur des ausführbaren Programms bezieht (genauer gesagt, die Festlegung der Segmente, auf die sich die berechneten *Offseten* beziehen), wird **zum Zeitpunkt der Verknüpfung** (*linking time*) **berechnet**. Die tatsächliche physikalische Adresse wird **beim Laden des Programms zur Ausführung berechnet** (*loading time*). Die tatsächliche Adresse wird immer an ein Segmentregister gemeldet. Dieses Register kann explizit angegeben werden, andernfalls

contrar, se asociază de către asamblor în mod implicit un registru de segment. Regulile pentru asocierea implicită sunt:

- CS** pentru etichete de cod destinație ale unor salturi (**jmp**, **call**, **ret**, **jz** etc);
- SS** în adresări SIB ce folosesc **EBP** sau **ESP** drept bază (indiferent de index sau scală);
- DS** pentru restul accesărilor de date.

Specificarea explicită a unui registru de segment se face cu ajutorul operatorului de prefixare segment (notat „::” și care se mai numește, ‘operatorul de specificare a segmentului’).

Observație: Dacă este omisă eticheta din adresarea directă folosită cu un index constant (de exemplu omiterea etichetei *table* din exprimarea *table*[100h]), este necesară atunci specificarea unui segment. Deplasamentul operandului este considerat drept punctul de început al segmentului specificat (care trebuie să aibă aceeași valoare cu deplasamentul etichetei *table* în cazul nostru) plus deplasamentul indexat. De exemplu, DS:[100h] reprezintă valoarea de la adresa 100h din segmentul referit de DS, exprimare echivalentă cu DS:100h.

Dacă se omite specificarea segmentului, este folosită valoarea constantă (imediată) a operandului și nu valoarea pe care o indică. De exemplu, [100h] desemnează chiar valoarea 100h, și nu valoarea de la adresa 100h.

Cât privește *operanții cu adresare indirectă*, aceștia utilizează registri pentru a indica adrese din memorie. Deoarece valorile din registri se pot modifica la momentul execuției, adresarea indirectă este indicată pentru a opera în mod dinamic asupra datelor.

wird das Segment standardmäßig einem Segmentregister zugeordnet. Die Regeln für die Standardzuordnungen sind:

- CS** für Zielcode-Labels einiger Sprünge (**jmp**, **call**, **ret**, **jz** usw.);
- SS** in SIB-Adressen, die **EBP** oder **ESP** als Basis verwenden (unabhängig von Index oder Skala);
- DS** für den Rest der Datenzugriffe.

Die explizite Angabe eines Segmentregisters erfolgt mit Hilfe des Segmentpräfix-Operators (vermerkt mit „::“ und der auch als "Segmentspezifikationsoperator" bezeichnet wird).

Hinweis: Wenn das Label in der direkten Adresse, die mit einem konstanten Index verwendet wird, weggelassen wird (z. B. die Label *table* in der Ausdruck *table*[100h] weggelassen wird), muss ein Segment angegeben werden. Das *Offset* des Operanden wird als Startpunkt des angegebenen Segments (das in unserem Fall den gleichen Wert wie das *Offset* des Label *table* haben muss) plus der indizierten *Offset* betrachtet. Zum Beispiel stellt DS:[100h] den Wert an der Adresse 100h des Segments dar, auf das sich DS bezieht, ein Ausdruck, der DS:100h entspricht.

Wenn die Segmentangabe weggelassen wird, wird der konstante (unmittelbare) Wert des Operanden verwendet und nicht der angegebene Wert. Beispielsweise bezeichnet [100h] sogar den Wert 100h und nicht den Wert an der Adresse 100h.

Bei den Operanden für indirekte Adressen verwenden sie Register, um Adressen aus dem Speicher anzuzeigen. Da die Werte in den Registern zur Laufzeit geändert werden können, wird die indirekte Adressierung angegeben, um die Daten dynamisch zu verarbeiten.

Forma generală pentru accesarea indirectă a unui operand de memorie este dată de formula de calcul a offset-ului unui operand:

$$[\text{registru_de_bază}] + [\text{registru_index} \times \text{scală}] + [\text{constantă}] \\ ([\text{Basis_Register}] + [\text{Index_Register} \times \text{Skala}] + [\text{Konstante}])$$

Constanta este o expresie a cărei valoare este determinabilă la momentul asamblării. De exemplu, $[\text{EBX} + \text{EDI} + \text{table} + 6]$ desemnează un operand prin adresare indirectă, unde atât *table* cât și 6 sunt constante.

Operanții *registru_de_bază* și *registru_index* sunt folosiți de obicei pentru a indica o adresă de memorie referitoare la un tablou. În combinație cu factorul de scalare, mecanismul este suficient de flexibil pentru a permite acces direct la elementele unui tablou de înregistrări, cu condiția ca dimensiunea în octeți a unei înregistrări să fie 1, 2, 4 sau 8. De exemplu, octetul superior al elementului de tip DWORD cu index dat în **ECX**, parte a unui vector de înregistrări al cărui adresă (a vectorului) este în **EDX** poate fi încărcat în **DH** prin intermediul instrucției

mov DH, [EDX + ECX * 4 + 3]

Din punct de vedere sintactic, când operandul nu este specificat prin formula completă (deoarece lipsesc unele dintre componente – de exemplu lipsește partea „* scală”), atunci asamblorul va rezolva ambiguitatea care rezultă analizând toate formele echivalente de codificare posibile și alegând-o pe cea mai scurtă dintre acestea. Cu alte cuvinte, având

push dword [EAX+EBX]; salvează pe stivă dublucuvântul de la adresa EAX + EBX

Die allgemeine Form für den indirekten Zugriff auf einen Speicheroperanden ergibt sich aus der Formel zur Berechnung des *Offsets* eines Operanden:

Die Konstante ist ein Ausdruck, dessen Wert zum Zeitpunkt der Assemblierung bestimmbar ist. Beispielsweise bezeichnet $[\text{EBX} + \text{EDI} + \text{table} + 6]$ einen Operanden durch indirekte Adressierung, wobei beide *table* und 6 konstant sind.

Die Operanden *Basis_Register* und *Index_Register* werden normalerweise verwendet, um eine Speicheradresse für eine Tabelle anzugeben. In Kombination mit dem Skalierungsfaktor ist der Mechanismus flexibel genug, um einen direkten Zugriff auf die Elemente eines Aufzeichnungstabellen zu ermöglichen, vorausgesetzt, die Bytegröße einer Aufzeichnung beträgt 1, 2, 4 oder 8. Zum Beispiel das obere Byte des Elements vom Typ DWORD mit in **ECX** angegebenem Index, Teil eines Datensatzvektors, dessen Adresse (des Vektors) in **EDX** ist, kann über die Anweisung

mov DH, [EDX + ECX * 4 + 3]
in **DH** hochgeladen werden.

Aus syntaktischer Sicht löst der Assembler die Mehrdeutigkeit, die sich aus einem Analyseprozess aller möglichen äquivalenten Codierungsformen und ergibt, auf, wenn der Operand nicht durch die vollständige Formel angegeben wird und einige der Komponenten fehlen (z. B. gibt es keine „* Skala“). Wählen Sie die kürzeste davon. Mit anderen Worten, mit

push dword [EAX + EBX]; Speichert das Doppelwort von der Adresse EAX + EBX auf dem Stapel

asamblorul are libertatea de a considera EAX drept bază și EBX drept index sau invers, EBX drept bază și EAX drept index.

Analog, pentru

pop dword [ECX]; restaurează vârful stivei în variabila cu adresa dată de ECX

asamblorul poate interpreta ECX fie ca bază fie ca index. Ce este realmente important de reținut este faptul că toate codificările luate în considerare de către asamblor sunt echivalente iar decizia finală a asamblorului nu are impact asupra funcționalității codului rezultat.

De asemenea, în plus față de rezolvarea unor astfel de ambiguități, asamblorul permite și exprimări non-standard cu condiția ca acestea să fie transformabile într-un final în forma standard de mai sus.

Alte exemple:

lea EAX, [EAX*2]; încarcă în EAX valoarea lui EAX*2 (adică, EAX devine 2*EAX)

În acest caz, asamblorul poate decide între codificare de tip bază = EAX + index = EAX și scală = 1 sau index = EAX și scală = 2.

lea EAX, [EAX*9 + 12]; EAX ia valoarea EAX * 9 + 12

Deși scală nu poate fi 9, asamblorul nu va emite aici un mesaj de eroare. Aceasta deoarece el va observa posibila codificare a adresei drept: bază = EAX + index = EAX cu scală = 8, unde de această dată valoarea 8 este corectă pentru scală. Evident, instrucțiunea putea fi precizată mai clar sub forma următoare:

der Assembler hat die Freiheit, EAX als Basis und EBX als Index oder umgekehrt, EBX als Basis und EAX als Index zu betrachten.

Analog, ist

pop Dword [ECX]; Stellt die Stapelspitze in der Variablen mit der von ECX angegebenen Adresse wieder her

der Assembler kann ECX entweder als Basis oder als Index interpretieren. Es ist wirklich wichtig, sich daran zu erinnern, dass alle vom Assembler berücksichtigten Codierungen gleichwertig sind und die endgültige Entscheidung des Assemblers keinen Einfluss auf die Funktionalität des resultierenden Codes hat.

Zusätzlich zum Auflösen solcher Mehrdeutigkeiten erlaubt der Assembler auch nicht standardmäßige Ausdrücke, vorausgesetzt, sie können endgültig in die obige Standardform umgewandelt werden.

Andere Beispiele:

lea EAX, [EAX * 2]; Ladet in EAX den Wert von EAX*2 (dh, EAX wird 2*EAX)

In diesem Fall kann der Assembler zwischen Basistypcodierung = EAX + Index = EAX und Skala = 1 oder Index = EAX und Skala = 2 wählen.

lea EAX, [EAX * 9 + 12]; EAX nimmt den Wert EAX * 9 + 12 an

Obwohl die Skala nicht 9 sein kann, gibt der Assembler hier keine Fehlermeldung aus. Dies liegt daran, dass er die mögliche Kodierung der Adresse als: base = EAX + index = EAX mit scale = 8 bemerkt, wobei diesmal der Wert 8 für die Skala korrekt ist. Offensichtlich könnte die Anweisung in der folgende Form klarer spezifiziert werden

lea EAX, [EAX + EAX * 8 + 12]

Să reținem deci că pentru adresarea indirectă, esențială este specificarea între paranteze drepte a cel puțin unuia dintre elementele componente ale formulei de calcul a *offset*-ului.

2.2 Utilizarea operatorilor

Operatorii se folosesc pentru combinarea, compararea, modificarea și analiza operanziilor. Unii operatori lucrează cu constante întregi, alții cu valori întregi memorate, iar alții cu ambele tipuri de operanzi.

Este importantă înțelegerea diferenței dintre operatori și instrucțiuni. Operatorii efectuează calcule cu valori constante determinabile la momentul asamblării. Instrucțiunile efectuează calcule cu valori ce pot fi necunoscute până în momentul execuției. Operatorul de adunare (+) efectuează adunarea în momentul asamblării; instrucțiunea ADD efectuează adunarea în timpul execuției.

Operatorii disponibili pentru construcția expresiilor sunt asemănători celor din limbajul C, atât ca sintaxă cât și din punct de vedere semantic. Evaluarea expresiilor numerice se face pe 64 de biți, rezultatele finale fiind ulterior ajustate în conformitate cu dimensiunea de reprezentare disponibilă în contextul de utilizare al expresiei.

În Tabel 1 sunt prezentate în ordinea priorității operatorii care pot fi folosiți în cadrul expresiilor limbajului de asamblare x86.

Daher ist zu beachten, dass für die indirekte Addressierung mindestens eines der Komponentenelemente der Offsetberechnungsformel in Klammern angegeben werden muss.

2.2 Verwendung von Operatoren

Operatoren werden zum Kombinieren, Vergleichen, Ändern und Analysieren von Operanden verwendet. Einige Operatoren arbeiten mit Ganzzahlkonstanten, andere mit gespeicherten Ganzzahlwerten und andere mit beiden Arten von Operanden.

Es ist wichtig, den Unterschied zwischen Operanden und Anweisungen zu verstehen. Die Operatoren führen Berechnungen mit konstanten Werten durch, die zum Zeitpunkt der Assemblierung ermittelt werden können. Die Anweisungen führen Berechnungen mit Werten durch, die bis zum Zeitpunkt der Ausführung unbekannt sein können. Der Additionoperator (+) führt die zum Zeitpunkt der Assemblierung durch. Die ADD-Anweisung führt die Addition während der Ausführung durch.

Die zur Erstellung von Ausdrücken verfügbaren Operatoren sind syntaktisch und semantisch denen in C-Sprache ähnlich. Die Auswertung der numerischen Ausdrücke erfolgt mit 64 Bit, wobei die Endergebnisse anschließend entsprechend der im Zusammenhang mit der Verwendung des Ausdrucks verfügbaren Darstellungsgröße angepasst werden.

In Tabelle 1 sind die Operatoren, die in den Ausdrücken der Assemblersprache x86 verwendet werden können, in der Reihenfolge ihrer Priorität aufgeführt.

Tabel 1. Operatorii care pot fi folosiți în cadrul expresiilor limbajului de asamblare x86.

Prioritate	Operator	Tip	Rezultat
7	-	Unar, präfix	Zweier Komplement (Negation): $-X = 0 - X$
7	+	Unar, präfix	Kein Effekt (angeboten für Symmetrie mit „-“): $+X = X$
7	~	Unar, präfix	Einer Komplement 1: mov AL, ~0 => mov AL, 0xFF
7	!	Unar, präfix	Logische Negation: $!X = 0$ wenn $X = 0$, andernfalls 1
6	*	Binär, infix	Multiplikation: $1 * 2 * 3 = 6$
6	/	Binär, infix	Der Betrag der Division ohne Vorzeichen: $24 / 4 / 2 = 3$ (-24/4/2 = 0FDh)
6	//	Binär, infix	Der Betrag der Division mit Vorzeichen: $-24 // 4 // 2 = -3$ (-24 / 4 / 2 ≠ -3!)
6	%	Binär, infix	Der Rest der Division ohne Vorzeichen: $123 \% 100 \% 5 = 3$
6	%%	Binär, infix	Der Rest der Division mit Vorzeichen: $-123 \% \% 100 \% \% 5 = -3$
5	+	Binär, infix	Aufsummierung: $1 + 2 = 3$
5	-	Binär, infix	Abnehmen: $1 - 2 = -1$
4	<<	Binär, infix	Bitweise Linksverschiebung: $1 << 4 = 16$
4	>>	Binär, infix	Bitweise Rechtsverschiebung: $0xFE >> 4 = 0x0F$
3	&	Binär, infix	UND: $0xF00F \& 0xFF6 = 0x0006$
2	^	Binär, infix	Exklusiv ODER: $0xFF0F ^ 0xFF = 0xFFFF$
1		Binär, infix	ODER: $1 2 = 3$

Operatorul de indexare are o utilizare largă în specificarea operanzilor din memorie adresați indirect. Rolul operatorului [] în adresarea indirectă a fost clarificat anterior.

2.2.1 Operatori de deplasare de biți

expresie >> cu_cât și expresie << cu_cât

Exemple:

mov AX, 01110111b << 3; desemnează valoarea 10111000b

add BX, 01110111b >> 3 ; desemnează valoarea 00001110b

2.2.2 Operatori logici pe biți

Operatorii pe biți efectuează operații logice la nivelul fiecărui bit al operandului (operanzilor) unei expresii. Expresiile au ca rezultat valori constante.

Der Indexierungsoperator wird häufig bei der Angabe von indirekt adressierten Speicheroperanden verwendet. Die Rolle des Operators [] bei der indirekten Adressierung wurde bereits erläutert.

2.2.1 Bitweise Verschiebungoperatoren

Ausdruck >> wie viel und Ausdruck << wie viel

Beispiele:

mov AX, 01110111b << 3; bezeichnet den Wert 10111000b

add BX, 01110111b >> 3; bezeichnet den Wert 00001110b

2.2.2 Bitweise Logischeoperatoren

Die Bitweise Operatoren führen logische Operationen auf der Ebene jedes Bits des Operanden eines Ausdrucks aus. Ausdrücke führen zu konstanten Werten.

OPERATOR	SYNTAX	SEMNIFICATIE (BEDEUTUNG)
\sim	$\sim \text{expresie}$	Bitweise Verneinung Ausdruck
$\&$	$\text{expr1} \& \text{expr2}$	Bitweise UND
$ $	$\text{expr1} \text{expr2}$	Bitweise ODER
\wedge	$\text{expr1} \wedge \text{expr2}$	Bitweise ODER EXKLUSIV

Exemple (presupunem că expresia se reprezintă pe un octet):

$\sim 11110000b$; rezultă valoarea $00001111b$

$01010101b \& 11110000b$; rezultă valoarea
 $01010000b$

$01010101b | 11110000b$; rezultă valoarea
 $11110101b$

$01010101b \wedge 11110000b$; rezultă valoarea
 $10100101b$

$!$ – negare logică (similar cu limbajul C) ; $!0 = 1$; $!(\text{orice diferit de zero}) = 0$

2.2.3 Operatorul de specificare a segmentului

Operatorul de specificare a segmentului ($:$) comandă calcularea adresei FAR a unei variabile sau etichete în funcție de un anumit segment. Sintaxa este:

segment:expresie

$[\text{SS:EBX+4}]$; deplasamentul e relativ la SS
 $[\text{ES:082h}]$; deplasamentul e relativ la ES
10h:var; segmentul este indicat de selectorul 10h, iar offset-ul este valoarea etichetei var.

2.2.4 Operatori de tip

Specifică tipurile unor expresii și a unor operanzi păstrați în memorie. Sintaxa pentru acestia este:

*tip expresie
(Typ Ausdruck)*

Beispiele (angenommen, der Ausdruck ist ein Byte):

$\sim 11110000B$; Der Wert 00001111b ergibt sich

$01010101b \& 11110000b$; es ergibt sich der Wert 01010000b

$01010101b | 11110000B$; es ergibt sich der Wert 11110101b

$01010101b \wedge 11110000b$; es ergibt sich der Wert 10100101b

$!$ - logische Verneinung (ähnlich der Sprache C); $!0 = 1$; $!(\text{etwas anderes als Null}) = 0$

2.2.3 Der Segmentspezifikationsoperator

Der Segmentspezifikationsoperator ($:$) befehlt die FAR-Adressberechnung einer Variablen oder eines Labels basierend auf einem bestimmten Segment. Die Syntax lautet:

Segment: Ausdruck

$[\text{SS:EBX+4}]$; Verschiebung ist relativ zu SS
 $[\text{ES:082h}]$; Verschiebung ist relativ zu ES
10h:var; Das Segment wird durch den Selektor 10h angezeigt, und der Offset ist der Wert des var-Labels.

2.2.4 Typoperatoren

Typoperatoren Sie geben die im Speicher abgelegten Arten von Ausdrücken und Operanden an. Die Syntax für sie lautet:

Această construcție sintactică forțează ca *expresia* să fie tratată considerându-se că are dimensiunea de reprezentare 10 *tip*, însă fără ca valoarea sa să fie modificată în mod definitiv (destructiv) în sensul precizat de conversia dorită. De aceea, aceștia sunt considerați *operatori de conversie (temporară) nedestructivă*. Pentru operanții păstrați în memorie, *tip* poate fi **BYTE**, **WORD**, **DWORD**, **QWORD** sau **TWORD** având dimensiunile de reprezentare 1, 2, 4, 8 și respectiv 10 octeți. Pentru etichetele de cod el poate fi **NEAR** (adresă pe 4 octeți) sau **FAR** (adresă pe 6 octeți).

Expresia **byte** [A] va indica doar primul octet de la adresa indicată de A. Analog, **dword** [A] indică dublucuvântul ce începe la adresa A.

Specificatorii BYTE / WORD / DWORD / QWORD au întotdeauna doar rol de a clarifica o ambiguitate (inclusiv când este vorba despre o variabilă de memorie, faptul de a preciza **mov BYTE [v], 0** sau **mov WORD [v], 0** este tot o clarificare a ambiguității, întrucât nasm nu asociază faptul ca v este byte/ word / dword).

În cazul instrucțiunii

mov [v], 0

se va primi mesajul „syntax error – operation size not specified”!

Specificatorul QWORD nu intervene niciodată explicit în cod pe 32 de biți.

Exemple unde e necesar un specificator de dimensiune al operanților:

- **mov [mem], 12**
- **(i)div [mem] ; (i)mul [mem]**
- **push [mem] ; pop [mem]**

Diese syntaktische Konstruktion erzwingt die Behandlung des Ausdrucks unter Berücksichtigung der Typdarstellungsdimension Typ, ohne dass sein Wert in dem durch die gewünschte Konvertierung angegebenen Sinne endgültig modifiziert (destructiv) wird. Sie gelten daher als *nicht destruktive (temporäre) Konvertierungsoperatoren*. Für im Speicher abgelegte Operanden kann der Typ **BYTE**, **WORD**, **DWORD**, **QWORD** oder **TWORD** mit den Darstellungsgrößen 1, 2, 4, 8 bzw. 10 Byte sein. Für Code-Labels kann **NEAR** (4-Byte-Adresse) oder **FAR** (6-Byte-Adresse) angegeben werden.

Das Ausdruck **byte** [A] gibt nur das erste Byte der durch A angegebenen Adresse an. Analog gibt **dword** [A] das Doppelwort an, das mit der Adresse A beginnt.

BYTE / WORD / DWORD / QWORD-Spezifizierer haben immer nur die Rolle, eine Mehrdeutigkeit zu klären (einschließlich der Angabe von **mov BYTE [v], 0** oder **mov WORD [v], 0** ist alles a Klarstellung der Mehrdeutigkeit, da nasm nicht die Tatsache assoziiert, dass v Byte / Wort / Dwort ist).

Im Falle von Anweisungen

mov [v], 0

Die Meldung „Syntaxfehler – Operationsgröße nicht angegeben“ („syntax error – operation size not specified“!) wird empfangen!

Der QWORD-Bezeichner greift niemals explizit in 32-Bit-Code **ein**.

Beispiele, bei denen ein Operandengrößenbezeichner erforderlich ist:

- **mov [mem], 12**
- **(i)div [mem]; (i)mul [mem]**
- **push [mem]; pop [mem]**

- **push 15** – aici este o inconsistență în NASM, asamblorul nu va emite eroare/warning ci va face **push DWORD 15**

Exemple de operanți IMPLICIȚI efectiv pe 64 biți (în cod pe 32):

mul dword [v]; înmulțește EAX cu dword-ul de la adresa v și depune rezultatul în perechea de registre EDX:EAX

div dword [v]; împărțire EDX:EAX la v

3. Directive

Directivele indică modul în care sunt generate codul și datele în momentul asamblării.

3.1 Directiva SEGMENT

Directiva SEGMENT permite direcționarea octetilor de cod sau date emisi de către un asamblor înspre segmentul precizat, segment care poartă un nume și are asociate diverse caracteristici.

SEGMENT nume [tip] [ALIGN = aliniere] [combinare] [utilizare] [CLASS = clasă]
(SEGMENT Name [Typ] [ALIGN = Ausrichtung] [Kombination] [Verwendung] [CLASS = Klasse])

Numelui segmentului i se asociază ca valoare adresa de segment (32 biți) corespunzătoare poziției segmentului în memorie în faza de execuție. În acest sens, asamblorul NASM pune la dispoziție și simbolul special \$\$ care este echivalent cu adresa segmentului curent, acesta având însă avantajul că poate fi utilizat în orice context, fără a fi necesar să fie cunoscut numele segmentului în care ne aflăm.

Cu excepția numelui, toate celelalte câmpuri sunt optionale din punct de vedere atât al prezenței, cât și al ordinii în care sunt

- **push 15** – Hier liegt eine Inkonsistenz in NASM vor. Die Assembler gibt keinen Fehler / keine Warnung aus, jedoch **push DWORD 15**

Beispiele für die impliziter Operanden 64-Bit effektiv (in 32-Bit-Code):

mul dword [v]; multiplizieren Sie EAX mit dem dword von Adresse v und senden Sie das Ergebnis in das EDX:EAX Registerpaar

div dword [v]; Division des EDX:EAX durch v

3. Direktiven

Die Direktiven geben an, wie der Code und die Daten zum Zeitpunkt der Assemblierung generiert werden.

3.1 Die SEGMENT- Direktive

Die SEGMENT-Direktive ermöglicht die Weiterleitung von Code- oder Datenbytes, die von einem Assembler ausgegeben werden, an das angegebene Segment, ein Segment, das einen Namen trägt und mit den verschiedenen Eigenschaften verknüpft sind.

Der Segmentname ist der Segmentadresse (32 Bit) zugeordnet, die der Position des Segments im Speicher in der Ausführungsphase entspricht. In dieser Hinsicht stellt die NASM-Assembler auch das spezielle Symbol \$\$ bereit, das ist äquivalent zu der aktuellen Segmentadresse. Dies hat jedoch den Vorteil, dass es in jedem Kontext verwendet werden kann, ohne den Namen des Segments zu kennen, in dem wir uns befinden.

Mit Ausnahme des Namens sind alle anderen Felder in Bezug auf Vorhandensein und Reihenfolge, in der sie angegeben sind,

specificate.

Argumentele optionale *tip*, *aliniere*, *combinare*, *utilizare* și 'clasa' dă editorului de legături și asamblorului indicații referitoare la modul de încărcare și atributelor segmentelor.

Tip permite selectarea unui model de folosire a segmentului, având la dispoziție următoarele opțiuni:

- **code** (sau **text**) – segmentul va conține cod, conținutul nu poate fi scris dar se poate citi sau executa;
- **data** (sau **bss**) – segment de date permitând citire și scriere, însă nu și execuție (valoare implicită);
- **rdata** – segment din care se poate doar citi, menit să conțină definiții de date constante

Argumentul optional *aliniere* specifică multiplul numărului de octeți la care trebuie să înceapă segmentul respectiv. Alinierile acceptate sunt puteri ale lui 2, între 1 și 4096.

Dacă argumentul *aliniere* lipsește, atunci se consideră implicit că este vorba despre o alinieră ALIGN = 1, adică segmentul poate începe la orice adresă.

Argumentul optional *combinare* controlează modul în care segmente cu același nume din cadrul altor module vor fi combinate cu segmentul în cauză la momentul editării de legături. Valorile posibile sunt:

- **PUBLIC** – indică editorului de legături să concateneze acest segment cu alte eventuale segmente cu același nume, obținându-se un unic segment a cărui lungime este suma lungimilor segmentelor componente.
- **COMMON** – specifică faptul că începutul acestui segment trebuie să se suprapună peste începutul tuturor

optional.

Optionale Argumente für *Typ*, *Ausrichtung*, *Kombination*, *Verwendung* und *Klasse* geben dem Link-Editor und dem Assembler Anweisungen zum Laden und Segmentieren von Attributen.

Mit **Typ** können wir ein Modell zur Verwendung des Segments auswählen. Folgende Optionen stehen zur Verfügung:

- **Code** (oder **Text**) – Das Segment enthält Code. Der Inhalt kann nicht geschrieben, aber gelesen oder ausgeführt werden;
- **data** (oder **bss**) – Datensegment zum Lesen und Schreiben, jedoch nicht zur Ausführung (Standard);
- **rdata** – Segment, aus dem Sie nur lesen können und das konstante Datendefinitionen enthalten soll.

Das optionale *Ausrichtungsargument* gibt das Vielfache der Anzahl der Bytes an, wo das Segment beginnen muss. Akzeptierte Ausrichtungen sind Zweierpotenzen zwischen 1 und 4096.

Wenn das *Ausrichtungsargument* fehlt, wird es standardmäßig als Alignment ALIGN = 1 betrachtet, dh das Segment kann an einer beliebigen Adresse beginnen.

Das optionale *Kombinationsargument* steuert, wie Segmente mit demselben Namen in anderen Modulen mit dem betreffenden Segment kombiniert werden, wenn Links bearbeitet werden. Mögliche Werte sind:

- **PUBLIC** – gibt dem Linkeditor an, dieses Segment mit anderen möglichen Segmenten desselben Namens zu verketten, um ein einzelnes Segment zu erhalten, dessen Länge die Summe der Längen der Komponentensegmente ist.
- **COMMON** – gibt an, dass der Anfang dieses Segments den Anfang aller gleichnamigen Segmente überlappen

segmentelor care au același nume. Se obține un segment având dimensiunea egală cu cea a celui mai mare segment având același nume.

- **PRIVATE** – indică editorului de legături că acest segment nu este permis a fi combinat cu altele care poartă același nume.
- **STACK** – segmentele cu același nume vor fi concatenate. În faza de execuție segmentul rezultat va fi segmentul stivă.

Implicit, dacă nu se specifică o metodă de combinare, orice segment este considerat **PUBLIC**.

Argumentul *utilizare (use)* permite optarea pentru altă dimensiune de cuvânt decât cea de 16 biți, care este implicită în lipsa precizării acestui argument.

Argumentul '*clasa*' are rolul de a permite stabilirea ordinii în care editorul de legături plasează segmentele în memorie. Toate segmentele având aceeași clasă vor fi plasate într-un bloc contigu de memorie indiferent de ordinea lor în cadrul codului sursă. Nu există o valoare implicită de inițializare pentru acest argument. Dacă nu este specificat, nu este definit. Drept consecință, se evită concatenarea într-un bloc continuu a tuturor segmentelor definite astfel.

```
segment code use32 class = CODE
```

3.2 Directiva ASSUME

Directiva **ASSUME** stabilește care sunt segmentele active la un moment dat. Ea are sintaxa generală:

```
ASSUME CS:nume1, SS:nume2, DS:nume3, ES:nume4
ASSUME CS:Name1, SS: Name2, DS: Name3, ES: Name4
```

unde ordinea specificărilor de după **ASSUME** nu este importantă, oricare și

muss. Man erhält ein Segment mit der Größe des größten gleichnamigen Segments.

- **PRIVATE** – zeigt dem Linkeditor an, dass dieses Segment nicht mit anderen Segmenten mit demselben Namen kombiniert werden darf.
- **STACK** – Segmente mit demselben Namen verkettet werden. In der Ausführungsphase ist das resultierende Segment das Stapelsegment.

Sofern keine Kombinationsmethode angegeben ist, wird jedes Segment standardmäßig als **PUBLIC** betrachtet.

Das Argument *Verwendung (use)* ermöglicht die Auswahl einer anderen Wortgröße als die 16-Bit-Dimension, was impliziert ist, wenn dieses Argument nicht angegeben wird.

Das Argument '*class*' soll die Reihenfolge zulassen, in der der Linkeditor die Segmente im Speicher ablegt. Alle Segmente mit derselben Klasse werden unabhängig von ihrer Reihenfolge im Quellcode in einem angrenzenden Speicherblock abgelegt. Es gibt keinen Standardinitialisierungswert für dieses Argument. Wenn nicht angegeben, ist es nicht definiert. Infolgedessen wird vermieden, alle so definierten Segmente in einem kontinuierlichen Block zu verketten.

```
segment data use32 class = DATA
```

3.2 Die ASSUME-Direktive

Die Direktive **ASSUME** bestimmt, welche Segmente zu einem bestimmten Zeitpunkt aktiv sind. Es hat die allgemeine Syntax:

Wenn die Reihenfolge der Spezifikationen nach **ASSUME** nicht wichtig ist, fehlen

oricăte dintre acestea putând să lipsească. Fiecare dintre *nume1*, *nume2*, *nume3* și *nume4* este un nume de segment sau cuvântul rezervat NOTHING.

Rolul directivei ASSUME este de a preciza asamblorului regăștrii de segment care trebuie utilizati pentru calculul adreselor efective ale etichetelor si ale variabilelor folosite în program. Existența acestei directive este necesară, deoarece un program poate fi alcătuit din mai multe segmente și este necesar ca asamblorul să cunoască în fiecare moment care sunt segmentele active.

Prefixarea explicită a etichetelor și variabilelor cu numele registrului de segment corespunzător (deci utilizarea operatorului de specificare a segmentului) furnizează în mod imediat această informație, având prioritate în cazul respectiv față de asocierea declarată prin directiva ASSUME.

Prezența acestei directive nu este necesară (având doar caracter de documentare) în cazul în care programul nu accesează etichete și variabile (astfel de programe sunt însă extrem de rare). De asemenea, dacă operanții în cadrul accesărilor indirecte nu fac referiri la etichete (de exemplu [EBX + 2] sau [EBP + EDI]) atunci se folosește ca registru de segment SS dacă apare EBP și respectiv EDS în celelalte cazuri. Aceste asocieri se fac automat, independent de ASSUME.

Este foarte important de reținut faptul că rolul acestei directive nu este și de a încărca registrele segment cu adresele corespunzătoare!

möglicherweise alle Angaben. Jeder der *Name1*, *Name2*, *Name3* und *Name4* ist ein Segmentname oder das reservierte Wort NOTHING.

Die Aufgabe der Direktive ASSUME besteht darin, die Segmentregister zu der Assembler anzugeben, die für die Berechnung der tatsächlichen Adressen der im Programm verwendeten Bezeichnungen und Variablen verwendet werden soll. Das Vorhandensein dieser Direktive ist notwendig, da ein Programm aus mehreren Segmenten bestehen kann und die Assembly zu jedem Zeitpunkt wissen muss, welche Segmente aktiv sind.

Das explizite Präfixieren von Labels und Variablen mit dem entsprechenden Segmentregisternamen (also unter Verwendung des Segmentspezifikationsoperators) stellt diese Informationen sofort zur Verfügung und hat in diesem Fall Vorrang vor der von der ASSUME-Direktive deklarierten Zuordnung.

Das Vorhandensein dieser Direktive ist nicht erforderlich (nur zu Dokumentationszwecken), wenn das Programm nicht auf Labels und Variablen zugreift (solche Programme sind jedoch äußerst selten). Wenn die Operatoren in den indirekten Zugriffen nicht auf Bezeichnungen verweisen (z. B. [EBX + 2] oder [EBP + EDI]), wird sie auch als SS-Segmentregister verwendet, wenn in den anderen Fällen EBP beziehungsweise EDS erscheinen. Diese Verknüpfungen werden unabhängig von ASSUME automatisch hergestellt.

Es ist sehr wichtig zu bedenken, dass die Rolle dieser Direktive nicht ist auch Segmentregister mit den entsprechenden Adressen zu laden!

3.3 Directive pentru definirea datelor

Definirea datelor înseamnă specificarea atributelor acestora și alocarea spațiului de memorie necesar. Deci definirea datelor este operația de *declarare* (specificarea atributelor) împreună cu cea de *alocare* (rezervarea a spațiului de memorie necesar).

În cadrul declarării datelor, principalul atribut specificat este *tipul datei* = *dimensiunea de reprezentare* – octet, cuvânt, dublucuvânt, quadword). Alocarea datelor se face în cadrul segmentului în care acestea au fost declarate.

Forma generală a unei linii sursă în cazul unei declarații de date este:

[*Name*] *Date_Typ Liste_der_Ausdrücke*
[*nume*] *tip_date lista_expresii*

oder

[*Name*] *Typ_Zuteilung Faktor*
[*nume*] *tip_alocare factor*

oder

[*Name*] **TIMES** *Faktor Date_Typ Liste_der_Ausdrücke* [;Kommentar]

unde *nume* este o etichetă prin care va fi referită data. Tipul rezultă din tipul datei (dimensiunea de reprezentare) iar valoarea este adresa la care se va găsi în memorie primul octet rezervat pentru data etichetată cu numele respectiv.

factor este un număr care indică de câte ori se repetă lista de expresii care urmează în paranteză.

Tip_data este o directive de definire a datelor, una din următoarele:

DB - date de tip octet (BYTE)

DW - date de tip cuvânt (WORD)

3.3 Diektiven zur Datendefinition

Das *Definieren* der Daten bedeutet, ihre Attribute anzugeben und den erforderlichen Speicherplatz zuzuweisen. Das Definieren der Daten erfolgt also durch die *Deklarationsoperation* (Angabe der Attribute) zusammen mit der *Zuordnungsoperation* (Reservierung des erforderlichen Speicherplatzes).

In der Datendeklaration wird als Hauptattribut *der Typ des Datums* angegeben = *die Dimension der Darstellung* (Byte, Wort, Doppelwort, Vierfachwort). Die Datenzuordnung erfolgt innerhalb des Segments, in dem sie deklariert wurden.

Die allgemeine Form einer Quellzeile bei einer Datendeklaration ist:

[;Kommentar]
[;comentariu]

[;Kommentar]
[;comentariu]

Wobei *Name* ein Label ist, die sich auf das Datum bezieht. Der Typ ergibt sich aus dem Datumstyp (Darstellungsgröße) und der Wert ist die Adresse, an der das erste Byte, das für das mit diesem Namen gekennzeichnete Datum reserviert ist, im Speicher abgelegt wird.

Faktor ist eine Zahl, die angibt, wie oft die Liste der Ausdrücke in Klammern wiederholt wird.

Date_Typ ist eine Datendefinitionsanweisung, eine der folgenden:
DB – Bytedaten (BYTE)

DD - date de tip dublucuvânt (DWORD)
DQ - date de tip 8 octeți (QWORD - 64 biți)
DT - date de tip 10 octeți (TWORD - utilizate pentru memorarea constantelor BCD sau constantelor reale de precizie extinsă).

De exemplu, secvența următoare definește și inițializează cinci variabile de memorie:

Data segment

```
var1    DB      'd'      ;1 octet
.a       DW      101b     ;2 octeți
var2    DD      2bfh     ;4 octeți
.a       DQ      307o     ;8 octeți (1 quadword)
.b       DT      100      ;10 octeți
```

Variabilele var1 și var2 sunt definite folosind etichete obișnuite, cu vizibilitate la nivelul întregului cod sursă, în timp ce .a și .b sunt etichete locale, accesul la aceste variabile impunând următoarele constrângeri:

- acestea se pot accesa cu numele local, adică .a sau .b, până în momentul definirii unei alte etichete obișnuite (ele fiind locale etichetei ce le preced);
- pot fi accesate de oriunde prin numele lor complet: var1.a, var2.a sau var2.b.

Valoarea de inițializare poate fi și o expresie, ca de exemplu în

vartest DW (1002/4+1)

După o directivă de definire a datelor pot să apară mai multe valori, permitându-se astfel declararea și inițializarea de tablouri. De exemplu, declarația

Tablou DW 1,2,3,4,5

creează un tablou de 5 întregi reprezentăți pe cuvinte având valorile respectiv 1, 2, 3, 4, 5. Dacă valorile de după directivă nu încap pe o

DW – Wortdaten (WORD)
DD – doppelwort Datentyp (DWORD)
DQ – Datentyp 8 Bytes (QWORD – 64 Bits)
DT – Datentyp 10 Bytes (TWORD – zum Speichern von BCD-Konstanten oder reellen Konstanten mit erweiterter Genauigkeit).

Die folgende Sequenz definiert und initialisiert beispielsweise fünf Speichervariablen:

Data segment

```
var1    DB      'd'      ;1 Byte
.a       DW      101b     ;2 Bytes
var2    DD      2bfh     ;4 Bytes
.a       DQ      307o     ;8 Bytes (1 quadword)
.b       DT      100      ;10 Bytes
```

Die Variablen var1 und var2 werden mit gemeinsamen Labels definiert, die im gesamten Quellcode sichtbar sind, während .a und .b lokale Labels sind. Der Zugriff auf diese Variablen unterliegt den folgenden Einschränkungen:

- Auf sie kann mit dem lokalen Namen, dh .a oder .b, bis zum Zeitpunkt der Definition eines anderen gemeinsamen Labels zugegriffen werden (sie sind lokal für das vorhergehende Label).
- können von überall mit ihrem vollständigen Namen abgerufen werden: var1.a, var2.a oder var2.b.

Der Initialisierungswert kann auch ein Ausdruck sein, z. B. in

vartest DW (1002/4 + 1)

Nach einer Direktive, die die Daten definiert, können mehrere Werte erscheinen, wodurch die Deklaration und Initialisierung von Tabellen ermöglicht wird. Zum Beispiel die Aussage
Tablou DW 1,2,3,4,5

erstellt eine Tabelle mit 5 Ganzzahlen, die durch Wörter mit den Werten 1, 2, 3, 4, 5 dargestellt werden. Wenn die Werte nach

singură linie se pot adăuga oricâte linii este necesar, linii ce vor conține numai directiva și valorile dorite. Exemplu:

Tabpatrate	DD	0, 1, 4, 9, 16, 25, 36
	DD	49, 64, 81
	DD	100, 121, 144, 169

Tip_alocare este o directivă de rezervare de date neinitializate:

RESB – date de tip octet (BYTE)
RESW – date de tip cuvânt (WORD)
RESD – date de tip dublucuvânt (DWORD)
RESQ – date de tip 8 octeți (QWORD – 64 biți)
REST – date de tip 10 octeți (TWORD – 80 biți)

factor este un număr care indică de câte ori se repetă tipul alocării precizate.

De exemplu:

Tabzero RESW 100h

rezervă 256 de cuvinte pentru tabloul *Tabzero*.

NASM nu suportă sintaxa din MASM/TASM pentru rezervarea de spațiu neinitializat scriind DW ? sau ceva similar. În NASM se procedează altfel: operandul unei pseudo-instrucțiuni de tip RESB este o **expresie critică** (toți operanții care intervin în calcul trebuie să fie cunoscuți în momentul în care expresia este întâlnită).

Exemplu:

buffer: **resb** 64; rezervă 64 octeți
wordvar: **resw** 1 ; rezervă un cuvânt
realarray **resq** 10; vector de 10 numere reale

Directiva TIMES permite asamblarea repetată a unei instrucțiuni sau definiții de

der Direktive nicht in eine einzelne Zeile passen, können Sie so viele Zeilen wie nötig hinzufügen, die nur die Direktive und die gewünschten Werte enthalten. Beispiel:

Tabpatrate	DD	0, 1, 4, 9, 16, 25, 36
	DD	49, 64, 81
	DD	100, 121, 144, 169

Typ_Zuteilung ist eine nicht initialisierte Datenreservierungsanweisung:

RESB – Bytedaten (BYTE)
RESW – Worttypdaten (WORD)
RESD – doppelter Datentyp (DWORD)
RESQ – Datentyp 8 Bytes (QWORD - 64 Bits)
REST - Datentyp 10 Bytes (TWORD - 80 Bits)

Der *Faktor* ist eine Zahl, die angibt, wie oft der angegebene Zuordnungstyp wiederholt wird.

Zum Beispiel:

Tabzero RESW 100h

256 Wörter für das Tabzero-Gemälde.

Unterstützt NASM die MASM / TASM-Syntax zum Buchen von nicht initialisiertem Speicherplatz durch Schreiben von DW nicht? oder so ähnlich. In NASM ist die Vorgehensweise anders: Der Operand eines RESB-Pseudobefehls ist ein **kritischer Ausdruck** (alle in die Berechnung eingreifenden Operanden müssen zum Zeitpunkt des Auftretens des Ausdrucks bekannt sein).

Beispiel:

Buffer: **resb** 64; reserviert 64 Bytes
wordvar: **resw** 1; reserviere ein Wort
realarray **resq** 10; Vektor von 10 reellen Zahlen

Die TIMES-Direktive ermöglicht das wiederholte Zusammenstellen einer

dată:

Anweisung oder von Datumsdefinitionen:

TIMES Faktor Date_Typ Ausdruck (expresie)

De exemplu:

Zum Beispiel:

Tabchar **TIMES** 80 DB ‘a’

creează un tablou de 80 de octeți inițializați fiecare cu codul ASCII al caracterului 'a'.

erstellt ein Array von 80 Bytes, die jeweils mit dem ASCII-Code des Zeichens 'a' initialisiert werden.

matrice10x10 **times** 10*10 dd 0

va furniza 100 de dublucuvinte dispuse continuu în memorie începând de la adresa asociată etichetei *matrice10x10*.

TIMES poate fi aplicată și instrucțiunilor:

liefert 100 Doppelworte, die fortlaufend im Speicher angeordnet sind, beginnend mit der Adresse, die dem Label *matrice10x10* zugeordnet ist.

Die **TIMES**-Direktive kann auch auf die Anweisungen angewendet werden:

TIMES Faktor Anweisung

TIMES 32 **add** EAX, EDX;

ceea ce are ca efect $EAX = EAX + 32*EDX$.

das hat die wirkung $EAX = EAX + 32*EDX$.

3.4 Directiva EQU

Directiva EQU permite atribuirea, în faza de asamblare, unei valori numerice sau sir de caractere unei etichete fără alocarea de spațiu de memorie sau generare de octeți. Sintaxa directivei EQU este

3.4 Die EQU-Direktive

Die *EQU-Direktive* ermöglicht die Zuweisung eines numerischen Werts oder einer Zeichenfolge zu einem Label in der Assembler-Phase, ohne Speicherplatz zuzuweisen oder Bytes zu generieren. Die Syntax der EQU-Direktive lautet

Name **EQU** Ausdruck

Exemple:

END_OF_DATA	EQU	'!'
BUFFER_SIZE	EQU	1000h
INDEX_START	EQU	(1000/4 + 2)
VAR_CICLARE	EQU	i

Beispiele:

END_OF_DATA	EQU	'!'
BUFFER_SIZE	EQU	1000h
INDEX_START	EQU	(1000/4 + 2)
VAR_CICLARE	EQU	i

Prin utilizarea de astfel de echivalări textul sursă poate deveni mai lizibil. Se observă asemănarea etichetelor echivalente prin directiva EQU cu constantele din limbajele de programare de nivel înalt.

Expresia pentru echivalarea unei etichete definite prin directiva EQU poate conține la rândul ei etichete definite prin EQU:

```
TABLE_OFFSET  
INDEX_START  
DICTIONAR_STAR
```

Durch die Verwendung solcher Entsprechungen kann der Quelltext besser lesbar werden. Wir beobachten die Ähnlichkeit der durch die EQU-Direktive äquivalenten Bezeichnungen mit den Konstanten der höheren Programmiersprachen.

Der Ausdruck für die Gleichwertigkeit eines in der EQU-Richtlinie definierten Label kann auch in der EQU definierte Labels enthalten:

```
EQU 1000h  
EQU (TABLE_OFFSET + 2)  
EQU (TABLE_OFFSET + 100h)
```

3.5 Directivele LABEL și PROC

Directiva **LABEL** permite numirea unei locații fără alocarea de spațiu de memorie sau generare de octeți, precum și accesul la o dată utilizând alt tip decât cel cu care a fost definită data respectivă. Sintaxa este:

3.5 LABEL- und PROC-Direktiven

Die **LABEL-Direktive** ermöglicht die Benennung eines Speicherorts ohne Zuweisung von Speicherplatz oder Byte-Generierung sowie den Zugriff auf ein Datum mit einem anderen Typ als dem, mit dem das Datum definiert wurde. Die Syntax lautet:

Name **LABEL** *Typ*

unde *name* este un simbol care nu a fost definit anterior în textul sursă, iar *tip* descrie dimensiunea de interpretare a simbolului și dacă acesta se va referi la cod sau la date.

Numele primește ca valoare contorul de locații. Tip poate să fie una din următoarele:

BYTE	NEAR	FAR
WORD	QWORD	PROC
DWORD	TBYTE	UNKNOWN

Tipurile **BYTE**, **WORD**, **DWORD**, **QWORD** și **TBYTE** etichetează respectiv date de 1, 2, 4, 8 și 10 octeți. Iată un exemplu de inițializare a unei variabile de memorie ca pereche de octeți însă accesată ca și cuvânt:

Dabei ist *Name* ein Symbol, das zuvor nicht im Quelltext definiert wurde, und der *Typ* beschreibt die Interpretationsdimension des Symbols und ob es sich auf den Code oder die Daten bezieht.

Der Name erhält den Wert des Programmzählers. Der Typ kann einer der folgenden sein:

Die Typen **BYTE**, **WORD**, **DWORD**, **QWORD** und **TBYTE** kennzeichnen jeweils Daten von 1, 2, 4, 8 und 10 Bytes. Hier ist ein Beispiel für die Initialisierung einer Speichervariablen als Bytepaar, auf das jedoch als Wort zugegriffen wird:

```

data segment
...
    Varcuv LABEL WORD
    DB 1,2
...
data ends

```

Tipul UNKNOWN declară un tip necunoscut și este folosit atunci când se dorește să existe posibilitatea accesării unei variabile de memorie în mai multe moduri (se aseamănă cu tipul *void* din limbajul C). De exemplu, accesarea variabilei *tempvar* din secvența de mai jos uneori ca octet și alteori ca și cuvânt poate fi realizată prin declararea ei ca etichetă de tip UNKNOWN:

```

data segment
...
    tempvar LABEL UNKNOWN
    DB ?,?
...
data ends

```

O altă soluție pentru adresarea unei date cu un alt tip decât cel cu care a fost declarată este utilizarea operatorului de conversie **PTR**.

Noțiunea de subrutină oferă posibilitatea dezvoltării modulare a programelor, permitând concentrarea atenției asupra punctelor esențiale ale unui program, ignorându-se în acele locuri detaliile de implementare. De asemenea, prezența subruteinilor aduce avantajul reutilizării codului (deci obținerea în final a unui cod cât mai compact), o subrutină fiind scrisă o singură dată, putând fi însă apelată de oricâte ori și în orice punct al programului cu valori diferite ale eventualilor parametri.

```

code segment
...
    mov AX, Varcuv
...

```

Der UNKNOWN-Typ deklariert einen unbekannten Typ und wird verwendet, wenn auf eine Speichervariable auf verschiedene Arten zugegriffen werden soll (ähnelt dem *void*-Typ in der C-Sprache). Der Zugriff auf die Variable *tempvar* in der folgenden Reihenfolge kann beispielsweise manchmal als Byte und manchmal als Wort erfolgen, indem Sie sie als UNKNOWN-Tag deklarieren:

```

code segment
...
    mov tempvar, AX; verwenden als Wort
...
    add DL, tempvar; verwenden als Byte
...

```

Eine andere Lösung für die Adressierung von Daten eines anderen Typs als dem, mit dem sie deklariert wurden, ist die Verwendung des **PTR**-Konvertierungsoperators.

Der Begriff der Subroutine bietet die Möglichkeit der modularen Entwicklung von Programmen, die es ermöglicht, sich auf die wesentlichen Punkte eines Programms zu konzentrieren und die Einzelheiten der Implementierung an diesen Stellen zu ignorieren. Das Vorhandensein von Unterprogrammen bringt auch den Vorteil mit sich, dass der Code wiederverwendet wird (um einen möglichst kompakten Code zu erhalten), wobei ein Unterprogramm nur einmal geschrieben wird, aber mit unterschiedlichen Werten der möglichen

În limbajul de asamblare 8086 definirea unei subrutine începe cu o directivă **PROC**:

Parameter jederzeit und an jedem Punkt des Programms aufgerufen werden kann.
In der Assemblersprache 8086 beginnt die Definition eines Unterprogramms mit einer **PROC-Anweisung**:

<Prozedurname> **PROC** [*Aufruftyp*]

unde *nume_procedură* este o etichetă reprezentând numele procedurii, iar *tip_apel* este NEAR sau FAR. Dacă lipsește, atunci se consideră implicit NEAR. Procedura va fi NEAR dacă va fi apelată numai în cadrul segmentului de cod în care este definită. Procedura va fi FAR dacă va fi apelată și din alte segmente de cod.

Directiva **ENDP** marchează sfârșitul unei subrutine ce începe cu PROC. Sintaxa ei este:

Dabei ist *Prozedurname* ein Label, das den Prozedurnamen darstellt, und der *Aufruftyp* ist NEAR oder FAR. Wenn es fehlt, wird es standardmäßig als NEAR betrachtet. Die Prozedur ist NEAR, wenn sie nur innerhalb des Codesegments aufgerufen wird, in dem sie definiert ist. Die Prozedur ist FAR, wenn sie aus anderen Codesegmenten aufgerufen wird.

Die **ENDP-Direktive** markiert das Ende eines Unterprogramms, das mit PROC beginnt. Ihre Syntax lautet:

<Prozedurname> **ENDP**

între cele două directive se vor scrie instrucțiunile și directivele corpului procedurii.

Zwischen den beiden Direktiven werden die Anweisungen und Direktiven des Hauptteils des Prozedur geschrieben.

3.6 Blocuri repetitive

Un bloc repetitiv este o construcție prin care i se cere asamblorului să genereze în mod repetat o configurație de octeți. Principalele blocuri repetitive sunt **REPT**, **IRP** și **IRPC**.

Un bloc repetitiv delimitat de directivele **REPT** și **ENDM** are următoarea sintaxă de definire:

3.6 Wiederholungsblöcke

Ein repetitiver Block ist ein Konstrukt, bei dem das Assembler wiederholt eine Bytekonfiguration generieren muss. Die Hauptwiederholungsblöcke sind **REPT**, **IRP** und **IRPC**.

Ein sich wiederholender Block, der durch die Anweisungen **REPT** und **ENDM** begrenzt ist, hat die folgende Definitionssyntax:

REPT *Zähler* (contor)
 Sequenz (secvență)

ENDM

Dies bedeutet, dass die Sequenz *Zähler* malen assembliert wird. Zum Beispiel die Sequenzen:

cu semnificația că secvența va fi asamblată de *contor* ori. De exemplu, secvențele:

```

Rept 5
    Dw    0           und
endm                                dw    0
                                         dw    0
                                         dw    0
                                         dw    0
                                         dw    0
                                         dw    0

```

generează același cod, lucru ce se poate realiza și cu: dw 5 DUP (0)

Exemplul următor însă nu are un echivalent atât de simplu. El generează 5 locații de memorie consecutive conținând valorile de la 0 la 4. Folosim în acest scop și directiva :=:

```

Intval = 0
Rept 10
    Dw    Intval      generieren werden
    Intval = Intval + 1
endm

```

Blocurile repetitive pot fi imbricate. Secvența :

dw	0
dw	1
dw	2
dw	3
dw	4

erzeugt den gleichen Code, der gemacht werden kann mit: dw 5 DUP (0)

Das folgende Beispiel hat jedoch kein so einfaches Äquivalent. Es erzeugt 5 aufeinanderfolgende Speicherplätze mit Werten von 0 bis 4. Wir verwenden auch die Direktive :=:

Wiederholte Blöcke können verschachtelt werden. Die Sequenz:

```

REPT 5
REPT 2
    Sequenz
ENDM
ENDM

```

generează de 10 ori secvența specificată.

Erzeugt 10 malen der angegebenen Sequenz.

Directiva **IRP** are sintaxa:

Die **IRP**-Direktive hat die Syntax:

```

IRP
    Parameter, <arg1 [arg2]...>
    Sequenz
ENDM

```

Se efectuează în mod repetat asamblarea secvenței, câte o dată pentru fiecare argument prevăzut în lista de argumente, prin înlocuirea textuală în secvență a

Die Assemblierung der Sequenz wird einmal für jedes in der Argumentliste angegebene Argument wiederholt, indem jedes Vorkommen des Parameters durch das

fiecărei apariții a parametrului cu argumentul curent. Argumentele pot fi siruri de caractere, simboluri, valori numerice. De exemplu:

IRP				
	param, <0, 1, 4, 9, 16, 25>		db	0
	db param	generieren werden	db	1
ENDM		die Sequenz	db	4
			db	9
			db	16
			db	25

Iar

und

IRP				
	reg,<AX,BX,CX,DX>	generieren werden	mov AX,DI	
	mov reg, DI	die Sequenz	mov BX,DI	
ENDM			mov CX,DI	
			mov DX,DI	

Directiva **IRPC** are un efect similar, ea realizând însă înlocuirea textuală a parametrului, pe rând, cu fiecare caracter dintr-un sir de caractere dat. Sintaxa ei este:

Die **IRPC**-Direktive hat eine ähnliche Wirkung, führt jedoch die textuelle Ersetzung des Parameters nacheinander durch jedes Zeichen in einer bestimmten Zeichenfolge durch. Ihre Syntax lautet:

De exemplu:

IRPC			
	<i>Parameter, Zeichenfolge</i>		
	<i>Sequenz</i>		
ENDM		Zu Beispiel	

IRPC	nr,1375
db	nr
ENDM	

creează 4 octeți având respectiv valorile 1, 3, 7 și 5.

Erstellt 4 Bytes mit den Werten 1, 3, 7 und 5.

3.7 Directiva INCLUDE

Directiva **INCLUDE** are sintaxa

3.7 Die INCLUDE-Direktive

Die **INCLUDE**-Direktive hat die folgende Syntax:

INCLUDE *Name_Datei*

Efectul ei (similar cu cel al directivei `#include` a preprocesorului C) este de a insera textual fișierul *nume_fișier* în textul sursă curent. Inserarea se face în locul în care apare directiva INCLUDE respectivă. *nume_fișier* este specificarea unui nume de fișier DOS, putând aşadar să conțină specificări de unitate de disc, directoare, nume de fișier și tip. În lipsa specificării unui tip se consideră implicit .ASM. Exemplu:

iar fișierul *instr2.asm* conține codul

```
cod segment
    mov ax,1
    INCLUDE INSTR2.ASM
    push AX
    mov BX,3
    add AX,BX
    dec BX
```

Rezultatul asamblării fișierului *prog.asm* va fi echivalent cu cel al asamblării codului

Das Ergebnis der Assemblierung der Datei *prog.asm* entspricht dem Assemblierung von der Code

```
cod segment
    mov ax,1
    mov BX,3
    add AX,BX
    dec BX
    push AX
```

Fișierele incluse pot conține la rândul lor alte directive INCLUDE, și.a.m.d. până la orice nivel, instrucțiunile respective fiind inserate corespunzător pentru crearea în final a unui singur cod sursă.

Seine Wirkung (ähnlich der der `#include`-Direktive von der Präprozessor C) besteht darin, den *Name_Datei* Datei textuell in den aktuellen Quelltext einzufügen. Das Einfügen erfolgt an der Stelle, an der die INCLUDE-Direktive erscheint. *Name_Datei* ist die Angabe eines DOS-Dateinamens, der Angaben zu Laufwerk, Verzeichnis, Dateiname und Typ enthalten kann. Wenn Sie keinen Typ angeben, wird standardmäßig .ASM berücksichtigt. Beispiel:

und die *instr2.asm*-Datei enthält den Code

Die enthaltenen Dateien können auch andere Direktiven wie INCLUDE usw. enthalten bis zu einer beliebigen Ebene, wobei die entsprechenden Anweisungen ordnungsgemäß für die endgültige Erstellung eines einzelnen Quellcodes eingefügt werden.

3.8 Macrouri

Un *macro* este un text parametrizat căruia își se atribuie un nume. Ori de câte ori găsește numele, assemblerul pune în codul sursă

3.8 Makros

Ein Makro ist ein parametrisierter Text, dem ein Name zugewiesen wird. Immer wenn der Name gefunden wird, fügt der Assembler

textul cu parametrii actualizați. Operația este cunoscută și sub numele de *expandarea macroului*.

Se poate face o analogie cu directiva INCLUDE prezentată anterior. Față de fișierele incluse, macrouile prezintă un grad sporit de flexibilitate permitând transmiterea de parametri și existența de etichete locale.

Un macro este delimitat de directivele **MACRO** și **ENDM** conform următoarei sintaxe:

```
Name MACRO  
[Parameter [.Parameter]...]  
Körper Anweisungen  
ENDM
```

De exemplu, pentru interschimbarea valorilor a două variabile cuvânt putem defini următorul macro:

```
swap MACRO a,b  
    mov ax, a  
    mov a,b  
    mov b,ax  
ENDM
```

Pentru înmulțirea cu 4 a valorii unei variabile (rezultatul depunându-se în DX:AX) se poate scrie următorul macro:

```
inmcu4 MACRO a  
    mov AX, a  
    sub DX, DX  
    shl AX,1  
    rcl DX, 1  
    shl AX,1  
    rcl DX, 1  
ENDM
```

Macrouile pot conține blocuri repetitive.

O posibilă problemă ce apare se referă la definirea unei etichete într-un macro. Să presupunem că se definește o etichetă și că în program există mai mult de un apel al

den Text mit den aktualisierten Parametern in den Quellcode ein. Die Operation wird auch als *Makroerweiterung* bezeichnet.

Eine Analogie kann mit der oben dargestellten INCLUDE-Direktive hergestellt werden. Makros sind im Vergleich zu den enthaltenen Dateien flexibler und ermöglichen die Übertragung von Parametern und das Vorhandensein lokaler Labels.

Ein Makro wird durch die Direktiven **MACRO** und **ENDM** gemäß der folgenden Syntax begrenzt:

Für den Werteaus tausch zweier Wortvariablen können wir beispielsweise folgendes Makro definieren:

Zum Multiplizieren des Werts einer Variablen mit 4 (das Ergebnis wird in DX:AX abgelegt) kann das folgende Makro geschrieben werden:

Makros können sich wiederholende Blöcke enthalten.

Ein mögliches Problem ist das Definieren eines Labels in einem Makro. Angenommen, ein Label ist definiert und es gibt mehr als einen Aufruf in diesem Makro im

acelui macro. Eticheta definită va apărea la fiecare expandare a macroului în codul programului, cauzând o eroare de „redefinire de etichetă”.

Soluția unei astfel de probleme este oferită de către directiva **LOCAL**, care, la apariția ei în cadrul unui macro, forțează ca domeniul de vizibilitate al etichetelor specificate ca argumente să fie numai acel macro.

Utilizarea directivei LOCAL trebuie să urmeze imediat directivei MACRO. Numărul argumentelor nu este limitat.

În cadrul utilizării macrourilor, referințele anticipate (*forward references*) nu sunt permise. Macrourile trebuie să fie definite întotdeauna înaintea invocării. De asemenea, macrourile pot fi imbicate.

Macrourile pot conține blocuri repetitive. De asemenea, macrourile pot invoca la rândul lor alte macrouri.

Programm. Das definierte Tag wird bei jeder Makroerweiterung im Programmcode angezeigt, was zu einem Fehler bei der Neudefinition dem Label führt.

Die Lösung eines solchen Problems bietet die **LOCAL**-Direktive, die in einem Makro das Sichtbarkeitsfeld der als Argumente angegebenen Labels auf dieses Makro beschränkt.

Die Verwendung der LOCAL-Direktive muss unmittelbar nach der MACRO-Direktive erfolgen. Die Anzahl der Argumente ist nicht begrenzt.

Bei der Verwendung von Makros sind Vorwärtsreferenz (*forward references*) nicht zulässig. Makros müssen immer vor dem Aufruf definiert werden. Makros können auch verschachtelt werden.

Makros können sich wiederholende Blöcke enthalten. Makros können auch andere Makros aufrufen.

Curs 7 și 8
Instrucțiuni ale limbajului de asamblare

Contents

1. Instrucțiuni de transfer al informației.....	4
1. Anweisungen zur Informationsübertragung	4
1.1 Instrucțiuni de transfer de uz general.....	4
1.1 TransferAnweisungen für allgemeine Zwecke	4
1.2 Instrucțiunea de transfer al adreselor LEA	8
1.2 Anweisung zur Übermittlung von LEA-Adressen	8
1.3 Instrucțiuni asupra flagurilor	9
1.3 Anweisungen auf Flaggen.....	9
1.4 Instrucțiuni de conversie (distructivă)	11
1.4 KonvertierungsAnweisungen (destruktiv)	11
1.5 Impactul reprezentării little-endian asupra accesării datelor	12
1.5 Auswirkung der Little-Endian-Darstellung auf den Datenzugriff	12
2. Operații.....	15
2. Operationen	15
2.1 Operații aritmetice	15
2.1 Rechenoperationen	15
2.1.1 Add, adc, sub, subc	16
2.1.1 Add, adc, sub, subc	16
2.1.2 Mul, imul	17
2.1.2 Mul, imul	17
2.1.3 Div, idiv	17
2.1.3 Div, idiv	17
2.1.4 Neg, inc, dec	17
2.1.4 Neg, inc, dec	17
2.2 Operații logice pe biți	18
2.2 Logische Operationen an Bits	18
2.3 Deplasări și rotiri de biți	18
2.3 Bewegungen und Drehungen	18
3. Ramificări, salturi, cicluri.....	19
3. Äste, Sprünge, Zyklen	19

3.1 Saltul necondiționat	19
3.1 Der bedingungslose Sprung	19
3.1.1 Instrucțiunea JMP	19
3.1.1 Die JMP-Anweisung	19
3.1.2 Instrucțiuni de salt condiționat și necondiționat	21
3.1.2 Bedingte und unbedingte SprungAnweisungen.....	21
3.1.3 Comparații între operanzi	23
3.1.3 Vergleiche zwischen Operanden.....	23
3.1.4 Instrucțiuni de ciclare	24
3.1.4 SchleifAnweisungen.....	24
3.2 Instrucțiunile CALL și RET	25
3.2 CALL- und RET-Anweisungen	25
4. Instrucțiuni pe șiruri	26
4. String Anweisungen	26
4.1 Generalități privind instrucțiunile pe șiruri	26
4.2 Instrucțiuni pe șiruri pentru transferul de date	29
4.2 String-Anweisungen für die Datenübertragung	29
4.3 Instrucțiuni pe șiruri pentru consultarea și compararea datelor	31
4.4 Execuția repetată a unei instrucțiuni pe șiruri	33
4.4 Wiederholte Ausführung eines String-Anweisung.....	33
4.5 Utilizarea de operanzi pentru instrucțiuni pe șiruri	Error! Bookmark not defined.
4.5 Die Verwendung von Operanden für String-Anweisungen.....	Error! Bookmark not defined.

Prezentăm forma generală a unui program în NASM, însăchuită de un scurt exemplu:

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 și printf au proveniență 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 msvert.dll; printf este funcție standard C și se regăsește în biblioteca msver.dll ;(SO)

; printf ist eine Standard-C-Funktion und befindet sich in der Bibliothek msver.dll ;(Betriebssystem)

bits 32 ; solicităm asamblarea pentru un procesor X86 (pe 32 biți)
; 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 funcției printf (adresa sirului) pe ; stivă (aşa cere printf)

; wir übertragen den Funktionsparameter printf (String-Adresse) auf den Stapel (so fragt
; printf)

call [printf] ; printf este numele unei funcții (etichetă = adresă,
; trebuie indirectată cu [])

; printf ist der Name einer Funktion (Label = Adresse muss mit [] angesprochen werden)

; apel ExitProcess(0), 0 reprezentând „execuție 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. Instrucțiuni de transfer al informației

1.1 Instrucțiuni de transfer de uz general

1. Anweisungen zur Informationsübertragung

1.1 TransferAnweisungen für allgemeine Zwecke

Tabelul 1. Instrucțiuni de transfer de uz general (TransferAnweisungen für allgemeine Zwecke)

MOV d,s	$<d> \leftarrow <s>$ (b-b, w-w, d-d)	-
PUSH s	ESP = ESP - 4 și depune $<s>$ în stivă (s – dublucuvânt) und deponiere $<s>$ im Stapel (s - Doppelwort)	-
POP d	extrag elementul curent din stivă și 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	$<d> \leftrightarrow <s>$	-
[reg_segment] XLAT	AL $\leftarrow <DS:[EBX+AL]>$ sau (oder) AL $\leftarrow <\text{reg_segment}:[EBX+AL]>$	-
CMOVcc d, s	$<d> \leftarrow <s>$ dacă cc (cod condiție) este adevărat wenn cc (bedingungscode) wahr ist	-
PUSHA / PUSHAD	Depune EDI, ESI, EBP, ESP, EBX, EDX, ECX și EAX pe stivă <i>Legt EDI, ESI, EBP, ESP, EBX, EDX, ECX und EAX auf dem Stapel ab</i>	-
POPA / POPAD	Extrag EAX, ECX, EDX, EBX, ESP, EBP, ESI și EDI de pe stivă <i>Extrahiert EAX, ECX, EDX, EBX, ESP, EBP, ESI und EDI aus dem Stapel</i>	-
PUSHF	Depune EFlags pe stivă <i>Legt EFlags auf dem Stapel ab</i>	-
POPF	Extrag vârful stivei și îl depune în EFlags <i>Extrahiert die Spitze des Stapels und legen Sie ihn in EFlags</i>	-
SETcc d	$<d> \leftarrow 1$ dacă cc este adevărat, altfel $<d> \leftarrow 0$ $<d> \leftarrow 1$, wenn cc wahr ist, andernfalls $<d> \leftarrow 0$	-

Dacă operandul destinație al instrucțiunii **MOV** este unul dintre cele 6 registre de segment, atunci sursa trebuie să fie unul dintre cele opt registre generale de 16 biți ale UE sau o variabilă de memorie. Încărătorul de programe al sistemului de operare pre-înitializează în mod automat registrele de segment, iar schimbarea valorilor acestora, deși 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 adiționale).

Instrucțiunile **PUSH** și **POP** au sintaxa:

PUSH s și respectiv **POP d**

Operanții trebuie să fie reprezentați pe dublucuvânt, deoarece stiva este organizată pe dublucuvinte. Stiva crește de la adrese mari spre adrese mici, din 4 în 4 octeți, ESP punctând întotdeauna spre dublucuvântul din vârful stivei.

Funcționarea acestor instrucțiuni poate fi ilustrată prin intermediul unei secvențe echivalente de instrucțiuni **MOV** și **ADD** sau **SUB**:

push EAX \Leftrightarrow **sub ESP, 4**
mov [ESP], EAX

; pregătim (alocăm) spațiu pentru a stoca valoarea
; Wir bereiten Speicherplatz vor, um den Wert zu
; speichern
; stocăm valoarea în locația alocată
; Wir speichern den Wert an dem zugewiesenen Ort

pop EAX \Leftrightarrow **mov EAX, [ESP]**
add ESP, 4

; încarcăm în EAX valoarea din vârful stivei
; Wir laden in EAX den Wert von der Spitze des
; Stapels
; eliberăm locația
; Wir räumen den Ort

Aceste instrucțiuni permit doar depunerea

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 Doppelwörter 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:

Diese Anweisungen erlauben nur das

și extragerea de valori reprezentate pe cuvânt și dublucuvânt. Ca atare,

PUSH AL

nu reprezintă o instrucțiune validă (syntax error), deoarece operandul nu are voie să aibă o valoare pe octet. Pe de altă parte, secvența de instrucțiuni:

PUSH AX	; depunem AX (<i>wir hinterlegen AX</i>)
PUSH EBX	; depunem EBX (<i>wir hinterlegen EBX</i>)
POP ECX	; ECX \leftarrow dublucuvântul din vârful stivei (valoarea lui EBX) ; ECX \leftarrow das Doppelwort oben im Stapel (der Wert von EBX)
POP DX	; DX \leftarrow cuvântul rămas în stivă (deci valoarea lui AX) ; DX \leftarrow das Wort im Stapel (also der Wert von AX)

este corectă și echivalentă prin efect cu

MOV ECX, EBX
MOV DX, AX

Adițional 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 și nucleul și bibliotecile de sistem. Implicația acestei constrângeri este că o instrucțiune de forma **PUSH** operand₁₆ sau **POP** operand₁₆ (de exemplu **PUSH** word 10), deși este suportată de către procesor și asamblată cu succes de către asamblor, nu este recomandată, putând cauza ceea ce poartă numele de eroare de dezalinierie a stivei: stiva este corect aliniată dacă și numai dacă valoarea din registrul ESP este în permanență divizibilă cu 4!

Instrucțiunea **XCHG** permite interschimbarea conținutului a doi operanzi de aceeași dimensiune (octet, cuvânt sau dublucuvânt), cel puțin unul dintre ei trebuind să fie registru. Sintaxa ei este:

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 (<i>wir hinterlegen AX</i>)
PUSH EBX	; depunem EBX (<i>wir hinterlegen EBX</i>)
POP ECX	; ECX \leftarrow dublucuvântul din vârful stivei (valoarea lui EBX) ; ECX \leftarrow das Doppelwort oben im Stapel (der Wert von EBX)
POP DX	; DX \leftarrow cuvântul rămas în stivă (deci valoarea lui AX) ; DX \leftarrow das Wort im Stapel (also der Wert von AX)

ist korrekt und äquivalent zu

MOV ECX, EBX
MOV DX, AX

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 Doppeltwörter oder Doppeltwörter 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 Doppelwort), 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 translatăre*. Instrucțiunea are sintaxa

[**reg_segment**] **XLAT**

tabelă_de_translatare este adresa directă a unui sir de octeți. Instrucțiunea **XLAT** pretinde la intrare adresa fară a tabelei de translatare furnizată sub unul din următoarele două moduri:

- DS:EBX (implicit, dacă lipsește precizarea registrului segment)
- **registru_segment:EBX**, dacă registrul segment este precizat explicit.

Efectul instrucțiunii **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, secvența:

```
    mov  EBX, Tabela  
    mov  AL, 6  
    ES xlat ; AL ← <ES:[EBX+6]>
```

depune conținutul celei de-a 7-a locații de memorie (de index 6) din *Tabelă* în AL.

Dăm un exemplu de secvență care translatează o valoare zecimală 'număr' cuprinsă între 0 și 15 în cifra hexazecimală (codul ei ASCII) corespunzătoare:

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 *Übersetzungstabellen* bezeichnet wird. Die Anweisung hat die Syntax

[**reg_segment**] **XLAT**

Übersetzungstabellen 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:

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 și se dovedește 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 Instrucțiunea de transfer al adreselor LEA

1.2 Anweisung zur Übermittlung von Adressen: LEA

LEA reg_general, mem	reg_general ← offset(mem)	-
-----------------------------	---------------------------	---

Instrucțiunea **LEA** (*Load Effective Address*) transferă deplasamentul operandului din memorie *mem* în registrul destinație. De exemplu:

lea EAX, [v]

încarcă în EAX offset-ul variabilei v, instrucțiune echivalentă cu:

mov EAX, v

Instrucțiunea **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, instrucțiunea:

lea EAX, [EBX + v - 6]

Die **LEA**-Anweisung (*Load Effective Address*) überträgt das Offset des Operanden aus dem *Mem*-Speicher in das Zielregister. Zum Beispiel:

lea EAX, [v]

lädt 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:

nu are ca echivalent direct o singură instrucțiune **MOV**, instrucțiunea

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 evidențiază prin versatilitate și eficiență sporite. Instrucțiunea **LEA** este considerată versatilă datorită faptului că permite combinarea unei înmulțiri cu adunări de valori stocate în registri și/sau cu valori constante. Pe de altă parte, instrucțiunea **LEA** este considerată eficientă datorită faptului că întregul calcul se execută într-o singură instrucțiune, fără a ocupa circuitele ALU care rămân astfel disponibile pentru alte operații (timp în care calculul de adresă este efectuat de către circuite specializate, separate, care fac parte din BIU).

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
```

lea eax, [ebx+8]
Put [ebx+8] into EAX. After this instruction, EAX will equal 0x00403A48.
Note
In contrast, the instruction **mov eax, [ebx+8]** will make EAX equal to 0x0012C140

Registers	Memory
EAX = 0x00000000	0x00403A40 0x7C81776F
EBX = 0x00403A40	0x00403A44 0x7C911000
	0x00403A48 0x0012C140
	0x00403A4C 0x7FFD8000

1.3 Instrucțiuni asupra flagurilor

Următoarele patru instrucțiuni sunt

1.3 Anweisungen auf Flaggen

Die folgenden vier Anweisungen sind

instrucțiuni de transfer al indicatorilor:

Instrucțiunea **LAHF** (*Load register AH from Flags*) copiază indicatorii SF (*Sign Flag*), ZF (*Zero Flag*), AF (*Adjust Flag*), PF (*Parity Flag*) și CF (*Carry Flag*) din registrul de flag-uri în biții 7, 6, 4, 2 și respectiv 0 ai registrului AH. Conținutul biților 5, 3 și 1 este nedefinit. Indicatorii nu sunt afectați în urma acestei operații de transfer (în sensul că instrucțiunea LAHF nu este ea însăși generatoare de efecte asupra unor flag-uri – ea doar copiază valorile flag-urilor).

Instrucțiunea **SAHF** (*Store register AH into Flags*) transferă biții 7, 6, 4, 2 și 0 ai registrului AH în indicatorii SF, ZF, AF, PF și respectiv CF, înlocuind valorile anterioare ale acestor indicatori.

Instrucțiunea **PUSHF** transferă toți indicatorii în vârful stivei (conținutul registrului Flags se transferă în vârful stivei). Indicatorii nu sunt afectați în urma acestei operații. Instrucțiunea **POPF** extrage cuvântul din vârful stivei și transferă din acesta indicatorii corespunzători în registrul de flag-uri.

Limbajul de asamblare pune la dispoziția programatorului niște *instrucțiuni de setare* a valorii indicatorilor de condiție, pentru ca programatorul să poată influența după dorință modul de acțiune al instrucțiunilor care exploatează *flag-uri*.

Anweisungen zum Übertragen der Indikatoren:

Die Anweisung **LAHF** (*Lade Register AH von Flags*) 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 (<i>Carry Flag</i>)
CMC	CF = ~CF	CF (<i>Carry Flag</i>)
STC	CF = 1	CF (<i>Carry Flag</i>)
CLD	DF = 0	DF (<i>Direction Flag</i>)
STD	DF = 1	DF (<i>Direction Flag</i>)

1.4 Instrucțiuni de conversie (distructivă)

1.4 Konvertierungsanweisungen (destruktiv)

CBW	conversie octet conținut în AL la cuvânt în AX (extensie de semn) <i>Byte-Inhalt von AL nach Wort nach AX konvertieren (Vorzeichenerweiterung)</i>	-
CWD	conversie cuvânt conținut în AX la dublu cuvânt în DX:AX (extensie de semn) <i>Konvertieren von Wortinhalten in AX in Doppelwort in DX:AX (Vorzeichenerweiterung)</i>	-
CWDE	conversie cuvânt din AX în dublucuvânt în EAX (extensie de semn) <i>Konvertieren dem Wort von AX in Doppelwort in EAX (Zeichenerweiterung)</i>	-
MOVZX d, s	încarcă în d (REGISTRU !), de dimensiune mai mare decât s, conținutul lui s fără semn <i>Ladet in d (REGISTER!), größer als s, den Inhalt von s ohne Vorzeichen</i>	-
MOVSX d, s	încarcă în d (REGISTRU !), de dimensiune mai mare decât s, conținutul lui s cu semn <i>Ladet in d (REGISTER!), größer als s, den Inhalt von s mit Vorzeichen</i>	-

Instrucțiunea **CBW** convertește 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 conținutul registrului AH). De exemplu:

```
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, instrucțiunea **CWD** extinde cuvântul cu semn din AX în dublucuvântul cu semn DX:AX. Exemplu:

```
mov ax,-10000; AX = 0D8F0h
 cwd           ; obține valoarea (Holen den Wert) -10000 in
               ; DX:AX (DX = 0FFFFh ; AX = 0D8F0h)
 cwde         ; obține valoarea (Holen den Wert) -10000 in EAX
```

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:

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:

; (EAX = 0FFFFD8F0h)

Conversia fără semn se realizează prin zeroizarea 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 obține prin aplicarea instrucțiunii **MOVZX**).

De ce coexistă **CWD** cu **CWDE**? **CWD** trebuie să rămână din rațiuni de *backwards compatibility* și din rațiuni de funcționalitate a instrucțiunilor (**IMUL** și **IDIV**).

```
MOV AH, 0C8h  
MOVSX EBX, AH  
MOVSX AX, [v]  
MOVZX EDX, AH  
MOVZX EAX, [v]
```

; EBX = FFFFFFFC8h
; MOVSX AX, byte ptr DS:[offset v]
; EDX = 000000C8h
; syntax error – op.size not specified

Atenție ! NU sunt acceptate sintactic!

Warum koexistiert **CWD** mit **CWDE**? Die **CWD** muss aus Gründen der Abwärtskompatibilität (*backwards compatibility*) und aus Gründen der Funktionalität der Anweisungen (**IMUL** und **IDIV** erhalten bleiben.

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

Dacă programatorul utilizează datele consistent cu dimensiunea de reprezentare stabilită la definire (exemplu: accesarea octetilor drept octeți și nu drept secvențe de octeți interpretate ca și cuvinte sau dublucuvinte, accesarea de cuvinte ca și cuvinte și nu ca perechi de octeți, accesarea de dublucuvinte ca și dublucuvinte și nu ca secvențe de octeți sau de cuvinte) atunci instrucțiunile limbajului de asamblare vor ține cont în mod AUTOMAT de modalitatea de reprezentare little-endian. Ca urmare, dacă se respectă această condiție programatorul nu trebuie să intervină suplimentar în nici un fel pentru a asigura corectitudinea accesării și

1.5 Auswirkung der Little-Endian-Darstellung auf den Datenzugriff

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

manipulării datelor utilizate. Exemplu:

a db 'd', -25, 120
b dw -15642, 2ba5h
c dd 12345678h

...

mov AL, [a]	;se încarcă în AL codul ASCII al caracterului 'd' ; <i>Man ladet den ASCII-Code des Zeichens 'd' in AL</i>
mov BX, [b]	;se încarcă în BX valoarea -15642; ordinea octetilor în BX va fi însă inversată față de reprezentarea în memorie, deoarece numai reprezentarea <i>în memorie</i> folosește reprezentarea <i>little-endian!</i> În registri datele sunt memorate conform reprezentării structurale normale, echivalente unei reprezentări <i>big endian</i> . ; <i>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.</i>
mov EDX, [c]	;se încarcă în EDX valoarea dublucuvânt 12345678h ; <i>der Doppelwort 12345678h wird in EDX geladen</i>

Dacă însă se dorește accesarea sau interpretarea datelor sub o formă diferită față de modalitatea de definire atunci trebuie utilizate conversii explicite de tip. În momentul utilizării conversiilor explicite de tip programatorul trebuie să își asume însă întreaga responsabilitate a interpretării și accesării corecte a datelor. În astfel de situații programatorul este obligat să conștientizeze particularitățile de reprezentare little-endian (ordinea de plasare a octetilor în memorie) și să utilizeze modalități de accesare a datelor în conformitate cu aceasta (cartea de curs, pag. 120-122).

Weise eingreifen, um den korrekten Zugriff und die Manipulation der verwendeten Daten sicherzustellen. Beispiel:

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 octetii sunt plasați 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) și 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 [a+4]

; DX:=1122h deoarece b+2 = a+4 , în sensul că aceste expresii de tip pointer desemnează aceeași adresă și anume adresa octetului 22h.

; 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 instrucțiune 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 adresa 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 octetii 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. Operații

2.1 Operații aritmetice

Operanții sunt reprezentați în Complementul lui 2. Când realizează adunările și scăderile, microprocesorul „vede” doar configurații de biți și nu numere cu semn sau fără. Regulile de efectuare a adunării și scăderii presupun adunarea de configurații binare, fără a fi nevoie de a interpreta operanții drept cu semn sau fără semn anterior efectuării operației! Deci, la nivelul acestor instrucțiuni, interpretarea „cu semn” sau „fără semn” rămâne la latitudinea programatorului, nefiind nevoie de instrucțiuni separate pentru adunarea/scăderea cu semn față de adunarea/scăderea fără semn.

Adunarea și scăderea se efectuează întotdeauna la fel (adunând sau scăzând configurații binare) indiferent de semnul (interpretarea) acestor configurații! După cum vom vedea acest lucru nu este valabil și pentru înmulțire și împărțire. În cazul acestor operații trebuie să știm apriori dacă operanții vor fi interpretați drept cu semn sau fără semn. De exemplu, fie doi operanți A și B reprezentați fiecare pe câte un octet:

2. Operationen

2.1 Rechnenoperationen

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 și -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 și în interpretarea cu semn)
 (= 74 sowohl in der vorzeichenlosen als auch in der vorzeichendeutung)

Microprocesorul realizează adunarea $C = A + B$ și obține:

$C = E6h = 11100110b$ (= 230 în interpretarea fără semn și -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 obținut, atât în interpretarea cu semn cât și în cea fără semn.

Der Mikroprozessor realisiert die Addition $C = A + B$ und erhält:

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 - reg_{8/16/32}, mem_{8/16/32}
- src – reg_{8/16/32}, mem_{8/16/32}, dată_imediată
- operațiile aritmetice și logice afectează următorii indicatori de condiție: CF, AC, ZF, SF, OF, PF
- eventuala depășire a capacitatei se verifică de către programator (atenție la forma de reprezentare cu / fără semn): CF – depășire la operațiile fără semn; OF – depășire la operațiile cu semn.

- dest - reg_{8/16/32}, mem_{8/16/32}
- src – reg_{8/16/32}, mem_{8/16/32}, 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

Instrucția **mul** realizează înmulțirea numerelor întregi fără semn, iar instrucția **imul** realizează înmulțirea numerelor întregi cu semn.

mul <i>src</i>	; <i>acc</i> := <i>acc</i> _{LO} × <i>src</i>
imul <i>src</i>	; <i>acc</i> := <i>acc</i> _{LO} × <i>src</i>
imul <i>dest, src1, imm_src</i>	; <i>dest</i> := <i>src1</i> × <i>imm_src</i>
imul <i>dest, imm_src</i>	; <i>dest</i> := <i>dest</i> × <i>imm_src</i>
imul <i>dest, src</i>	; <i>dest</i> := <i>dest</i> × <i>src</i>

- *src* – reg_{8/16/32}, mem_{8/16/32}
- *acc* – AX, DX:AX, EDX:EAX ($\dim(\text{src}) \times 2$)
- *dest* – reg_{16/32}
- *src1* – reg_{16/32}, mem_{16/32}
- *imm_src* – data imediată_{8/16/32}
 - Sofort_Data_{8/16/32}

2.1.3 Div, idiv

Instrucția **div** realizează împărțirea numerelor întregi fără semn, iar instrucția **idiv** realizează împărțirea numerelor întregi cu semn.

div <i>src</i>	; <i>acc</i> _{LO} := <i>acc</i> / <i>src</i>
idiv <i>src</i>	; <i>acc</i> _{HI} := <i>acc</i> MOD <i>src</i> ; <i>acc</i> _{LO} := <i>acc</i> / <i>src</i> ; <i>acc</i> _{HI} := <i>acc</i> MOD <i>src</i>

- *src* – reg_{8/16/32}, mem_{8/16/32}
- *acc* – AX, DX:AX, EDX:EAX ($2 \times \dim(\text{src})$)

Nu se pot împărți 2 operanți de aceeași lungime! Indicatorii de condiție (*flags*) au conținut imprevizibil după înmulțire și împărțire!

2.1.4 Neg, inc, dec

Instrucția **neg** calculează opusul unui număr. Instrucția **inc** incrementează conținutul destinației, iar instrucția **dec**

2.1.2 Mul, imul

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 <i>src</i>	; <i>acc</i> := <i>acc</i> _{LO} × <i>src</i>
imul <i>src</i>	; <i>acc</i> := <i>acc</i> _{LO} × <i>src</i>
imul <i>dest, src1, imm_src</i>	; <i>dest</i> := <i>src1</i> × <i>imm_src</i>
imul <i>dest, imm_src</i>	; <i>dest</i> := <i>dest</i> × <i>imm_src</i>
imul <i>dest, src</i>	; <i>dest</i> := <i>dest</i> × <i>src</i>

- *src* – reg_{8/16/32}, mem_{8/16/32}
- *acc* – AX, DX:AX, EDX:EAX ($\dim(\text{src}) \times 2$)
- *dest* – reg_{16/32}
- *src1* – reg_{16/32}, mem_{16/32}
- *imm_src* – data imediată_{8/16/32}
 - Sofort_Data_{8/16/32}

2.1.3 Div, idiv

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 <i>src</i>	; <i>acc</i> _{LO} := <i>acc</i> / <i>src</i>
idiv <i>src</i>	; <i>acc</i> _{HI} := <i>acc</i> MOD <i>src</i> ; <i>acc</i> _{LO} := <i>acc</i> / <i>src</i> ; <i>acc</i> _{HI} := <i>acc</i> MOD <i>src</i>

- *src* – reg_{8/16/32}, mem_{8/16/32}
- *acc* – AX, DX:AX, EDX:EAX ($2 \times \dim(\text{src})$)

Sie können nicht 2 Operanden gleicher Länge teilen! Die Bedingungsindikatoren (Flags) enthielten nach Multiplikation und Division unvorhersehbaren Inhalt!

2.1.4 Neg, inc, dec

Die **neg**-Anweisung berechnet das Gegenteil einer Zahl. Der Anweisung **inc** erhöht den Inhalt des Ziels und der Anweisung **dec**

decrementează conținutul destinației.

neg <i>dest</i>	; <i>dest</i> := - <i>dest</i>
inc <i>dest</i>	; <i>dest</i> := <i>dest</i> + 1
dec <i>dest</i>	; <i>dest</i> := <i>dest</i> - 1

- *dest* – reg_{16/32}, mem_{16/32}

Acste instrucțiuni sunt scurte și rapide. Ele sunt utile pentru parcurgerea unor șiruri prin incrementarea sau decrementarea adresei. De asemenea, ele sunt utile pentru contorizare (numărare).

2.2 Operații logice pe biți

Instrucțiunea **AND** este indicată pentru izolare unui anumit bit sau pentru forțarea anumitor biți la valoarea 0.

Instrucțiunea **OR** este indicată pentru forțarea anumitor biți la valoarea 1.

Instrucțiunea **XOR** este indicată pentru schimbarea valorii unor biți din 0 în 1 sau din 1 în 0.

2.3 Deplasări și rotiri de biți

Instrucțiunile de *deplasare* de biți se clasifică în:

- Instrucțiuni de deplasare logică: stânga **SHL**, dreapta **SHR**
- Instrucțiuni de deplasare aritmetică: stânga **SAL**, dreapta **SAR**

Instrucțiunile de *rotire* a biților în cadrul unui operand se clasifică în:

- Instrucțiuni de rotire fără carry: stânga **ROL**, dreapta **ROR**
- Instrucțiuni de rotire cu carry: stânga **RCL**, dreapta **RCR**

Instrucțiunile de deplasare și rotire de biți – tabel – pag.134 curs.

Pentru a defini deplasările și rotirile să considerăm ca și configurație inițială un

verringert den Inhalt des Ziels.

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 Logische Operationen an Bits

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 Bewegungen und Drehungen

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**, Rechtlauf **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

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:

```
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

3.1 Saltul necondiționat

În această categorie intră instrucțiunile **JMP** (echivalentul instrucțiunii **GOTO** din alte limbaje), **CALL** (apelul de procedură înseamnă transferul controlului din punctul apelului la prima instrucțiune din procedura apelată) și **RET** (transfer control la prima instrucțiune executabilă de după **CALL**).

JMP operand ; Salt necondiționat la adresa determinată de operand

CALL operand; Transferă controlul procedurii determinată de operand

RET [n] ; Transferă controlul instrucțiunii de după **CALL**

3.1.1 Instrucțiunea JMP

Instrucțiunea de salt necondiționat **JMP** are sintaxa : **JMP operand** unde *operand* este o etichetă, un registru sau o variabilă de memorie care conține o adresă. Efectul ei este transferul necondiționat al controlului la instrucțiunea care urmează

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. Äste, Sprünge, Zyklen

3.1 Der bedingungslose Sprung

Diese Kategorie umfasst die Anweisung **JMP** (das Äquivalent der **GOTO**-Anweisung in anderen Sprachen), **CALL** (der Prozederaufruf 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 Die JMP-Anweisung

Der unbedingte JMP-Sprung-Anweisung 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

etichetei, la adresa dată de valoarea registrului sau constantei, respectiv la adresa conținută în variabila de memorie. De exemplu, după execuția secvenței

<i>AdunaUnu:</i>	mov AX, 1 jmp <i>AdunaDoi</i> inc EAX jmp urmare
<i>AdunaDoi:</i>	add EAX, 2
urmare:	.
	.

registru AX va conține valoarea 3. Instrucțiunile **inc** și **jmp** dintre etichetele *AdunaUnu* și *AdunaDoi* nu se vor executa, decât dacă se va face salt la *AdunaUnu* de altundeva din program.

Saltul poate fi făcut și la o adresă memorată într-un registru sau într-o variabilă de memorie. Exemple:

- (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 destinație registru cu un operand destinație variabilă de memorie, o soluție 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]
```

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:

<i>AdunaUnu:</i>	mov AX, 1 jmp <i>AdunaDoi</i> inc EAX jmp urmare (<i>Ergebnis</i>)
<i>AdunaDoi:</i>	add EAX, 2
<i>Ergebnis:</i>	.
	.

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:

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 Instrucțiuni de salt condiționat și necondiționat

Următoarele instrucțiuni au ca efect realizarea unui salt în cadrul programului doar dacă anumite condiții sunt îndeplinite.

Jcc - salt dacă condiția 'cc' este îndeplinită; în caz contrar se trece la instrucțiunea următoare

Syntax: **Jcc <eticheta>**, unde:

- <eticheta> - se traduce printr-o distanță relativă pe 8 biți
- condiția este dată de starea unui sau a unor indicatoare de condiție (*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 în intervalul -128 .. +127;
- De la '386 salturile se pot face oriunde în interiorul unui segment

3.1.2 Bedingte und unbedingte SprungAnweisungen

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)

Instrucțiune (Anweisung)	Condiție (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 = 1	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)

Instrucțiune (Anweisung)	Condiție (Bedingung)	Indicator	Alias
JA	>	CF = 0; ZF = 0	JNBE
JAE	\geq	CF = 0	JNB, JNC
JB	<	CF = 1	JNAE, JC
JBE	\leq	CF = 1 oder ZF = 1	JNA
JE	=	ZF = 1	JZ
JNE	\neq	ZF = 0	JNZ

Tabelul 4. Instrucțiuni de salt condiționat – comparare numere cu semn (Bedingte Sprunganweisungen – Vergleich von Zahlen mit Vorzeichen)

Instrucțiune (Anweisung)	Condiție (Bedingung)	Indicator	Alias
JG	>	SF = OF sau ZF = 0	JNLE
JGE	\geq	SF = OF	JNL
JL	<	SF \neq OF	JNGE
JLE	\leq	SF \neq OF sau ZF = 1	JNG
JE	=	ZF = 1	JZ
JNE	\neq	ZF = 0	JNZ

În Tabelul 2, Tabelul 3 și Tabelul 4 (vezi și tabelul de la pagina 146 din cursul tipărit) se prezintă instrucțiunile de salt condiționat împreună cu semnificația lor și cu precizarea valorilor *flag*-urilor în urma căror se execută salturile respective. Pentru toate instrucțiunile de salt sintaxa este aceeași:

<instrucțiune_de_salt> etichetă

Semnificația instrucțiunilor de salt condiționat este dată sub forma „*salt dacă operand1 <> relație> față de operand2*” (unde cei doi operanzi sunt obiectul unei instrucțiuni anterioare **CMP** sau **SUB**) sau referitor la valoarea concretă setată pentru un anumit *flag*.

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.

Când se compară două numere cu semn se folosesc termenii „**less than**” (mai mic decât) și „**greater than**” (mai mare decât), iar când se compară două numere fără semn se folosesc termenii „**below**” (inferior, sub) și respectiv „**above**” (superior, deasupra, peste).

3.1.3 Comparații între operanzi

Instrucțiunile de salt condiționat se folosesc de obicei în combinație cu instrucțiuni de comparare. De aceea, semnificațiile instrucțiunilor de salt rezultă din semnificația operanzilor unei instrucțiuni de comparare. În afara testului de egalitate pe care îl poate efectua o instrucțiune **CMP** este de multe ori necesară determinarea relației 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 și nu! Dacă cele două numere sunt considerate fără semn, atunci primul are valoarea 255 și este evident mai mare decât 0. Dacă cele două numere sunt considerate cu semn, atunci primul are valoarea -1 și este mai mic decât 0.

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 Vergleiche zwischen Operanden

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 <i>d,s</i>	comparația valorilor operanzilor (nu modifică operanzii) (execuție fictivă <i>d - s</i>) <i>Vergleich der Werte der Operanden (ändert die Operanden nicht) (fiktive Ausführung d - s, Subtraktion)</i>	OF, SF, ZF, AF, PF și CF
TEST <i>d,s</i>	execuție fictivă <i>d AND s</i> <i>(fiktive Ausführung d UND s)</i>	OF = 0, CF = 0 SF, ZF, PF – modificați (<i>modifizierten</i>), AF – nedefinit (<i>undefiniert</i>)

Instrucțiunea **CMP** nu face distincție între cele două situații, deoarece adunarea și scăderea se efectuează întotdeauna la fel (adunând sau scăzând configurații binare)

Die **CMP**-Anweisung unterscheidet nicht zwischen den beiden Situationen, da die Addition und Subtraktion unabhängig vom Vorzeichen (Interpretation) dieser

indiferent de semnul (interpretarea) acestor configurații. Ca urmare, nu este vorba de a interpreta cu semn sau fără semn *operanții* scăderii fictive *d-s*, ci *rezultatul* final al acesteia! Rolul de a interpreta în mod diferit (cu semn sau fără semn) rezultatul final al comparației revine diverselor instrucțiuni de salt condiționat.

3.1.4 Instrucțiuni de ciclare

Instrucțiunea **JCXZ (JECXZ)** permite realizarea unui salt dacă registrul CX (respectiv ECX) are valoarea 0. Ea se folosește înaintea unei instrucțiuni de buclare (**LOOP**) pentru a preîntâmpina execuția de 65.535 ori a buclei, în cazul în care CX=0.

O altă instrucțiune de buclare este **LOOP**, având sintaxa:

LOOP <eticheta>

În cadrul acestei instrucțiuni se efectuează:
ECX = ECX - 1
if (ECX != 0) „salt la <eticheta>”
else „continuă cu instrucțiunea următoare”
Saltul este „scurt” (max. 127 octeți – atenție deci la „distanța” dintre **LOOP** și etichetă!).

ECX este folosit implicit pentru contorizarea ciclurilor executate.

Dintre instrucțiunile de salt necondiționat, prezentăm **LOOPZ** și **LOOPE (LOOP while Equal)** – instrucțiuni de buclare.

Sintaxa:

LOOPZ | LOOPE <eticheta>

La fel ca în cazul instrucțiunii **LOOP**,

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 SchleifAnweisungen

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

acțiunea lor este următoarea:

ECX = ECX - 1

if ((ECX != 0) și (ZF = 1) „salt la eticheta“
else „continuă“

Alte instrucțiuni de buclare sunt **LOOPNZ** și **LOOPNE**. Sintaxa lor:

LOOPNZ | LOOPNE <eticheta>

Acțiunea lor:

ECX = ECX - 1

if ((ECX!=0) și (ZF!=1) „salt la eticheta“
else „continuă“

Nici una dintre instrucțiunile de ciclare prezentate nu afectează flag-urile.

3.2 Instrucțiunile CALL și RET

Apelul unei proceduri se face cu ajutorul instrucțiunii **CALL**, acesta putând fi *apel direct* sau *apel indirect*. Apelul direct are sintaxa:

CALL operand

Asemănător instrucțiunii **JMP** și instrucțiunea **CALL** transferă controlul la adresa desemnată de operand. În plus față de aceasta, înainte de a face saltul, instrucțiunea **CALL** salvează în stivă adresa următoarei instrucțiuni de după **CALL** (adresa de revenire). Cu alte cuvinte, avem echivalența:

CALL operand

A: . . .

↔

Terminarea execuției secvenței apelate este marcată de întâlnirea unei instrucțiuni **RET**. Aceasta preia din stivă adresa de revenire depusă acolo de **CALL**, predând controlul la instrucțiunea de la această adresă. Sintaxa instrucțiunii **RET** este:

RET [n]

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 CALL- und RET-Anweisungen

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:

push A

jmp operand

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]

unde n este un parametru optional. El indică eliberarea din stivă a n octeți aflați sub adresa de revenire.

Instrucțiunea **RET** poate fi ilustrată prin echivalență:

RET n (revenire (<i>Rückkehr</i>) near)	\Leftrightarrow	B dd ? . . pop [B] add ESP,[n] jmp [B]
---	-------------------	--

De cele mai multe ori, instrucțiunile **CALL** și **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*)

Instrucțiunea **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 funcției *printf*)

Rezumând, operandul destinație al unei instrucțiuni **CALL** poate fi:

- numele unei proceduri
- numele unui registru în care se află o adresă
- o adresă de memorie

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:

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 Zielperand einer **CALL**-Anweisung sein:

- der Name einer Prozedur
- der Name eines Registers, in dem sich eine Adresse befindet
- eine Speicheradresse

4. Instrucțiuni pe şiruri

4.1 Generalități privind instrucțiunile pe şiruri

Instrucțiunile pe şiruri permit manipularea unui bloc de date printr-o singură

4. String Anweisungen

4.1 Allgemeines zu String-Anweisungen

String-Anweisungen ermöglichen die Manipulation eines Datenblocks durch eine

instrucțiune. Sunt singurele instrucțiuni care permit transfer memorie-memorie sau memorie-port de intrare / ieșire.

Instrucțiunile pe șiruri sunt puternice, ele având ca efect simultan (eventual repetat) atât accesarea memoriei cât și incrementarea sau decrementarea unor registre pointer. Se folosesc pentru manipularea șirurilor de octeți sau cuvinte, fiind mai scurte (ca și cod rezultat) și mai rapide în execuție decât combinațiile echivalente de instrucțiuni **MOV**, **INC** și **LOOP**.

Instrucțiunile pe șiruri folosesc operanzi implicați:

- DS:ESI – adresa elementului din șirul sursă
- ES:EDI – adresa elementului din șirul destinație
- ECX – contor
- AL/AX – registru acumulator
- incrementarea sau decrementarea automată a regisrelor index (ESI, EDI) în funcție de starea *flag*-ului DF (*Direction Flag*) (0 = incrementare, 1 = decrementare)
- decrementarea regisrului ECX

Un șir în sensul 8086 este caracterizat de următoarele atribute:

- a) tipul elementelor (octeți sau cuvinte).
- b) adresa primului element din șir.
- c) direcția de parcurgere (de la adrese mici spre adrese mari sau invers).
- d) numărul de elemente.

Din punct de vedere al instrucțiunilor pe șiruri, un șir poate fi șir sursă sau șir destinație.

Instrucțiunile pe șiruri sunt în număr de 10 și se împart în trei categorii:

- instrucțiuni care folosesc un șir sursă și

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:

- a) die Art der Elemente (Bytes oder Wörter).
- b) die Adresse des ersten Elements in der Zeichenfolge.
- c) die Fahrtrichtung (von kleinen Adressen zu großen Adressen oder umgekehrt).
- d) 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

un sir destinație (**MOVSB**, **MOVSW**, **CMPSB**, **CMPSW**).

- instrucțiuni care folosesc numai un sir sursă (**LODSB**, **LODSW**).
- instrucțiuni care folosesc numai un sir destinație (**STOSB**, **STOSW**, **SCASB**, **SCASW**).

Fiecare dintre aceste instrucțiuni pretinde să-i fie pregătite în prealabil caracteristicile sirurilor cu care operează.

a) *Tipul* este indicat prin ultima literă a numelui instrucțiunii: B pentru elemente octeți și W pentru elemente cuvinte. Instrucțiunile care folosesc două siruri presupun că ambele siruri sunt de același tip.

b) *Adresa primului element dintr-un sir* este o adresă FAR memorată în regiștri, astfel:

- în DS:SI pentru sirurile care sunt sursă;
- în ES:DI pentru sirurile care sunt destinație;
- c) Direcția de parcursere este indicată de flagul DF, astfel:

- DF = 0 impune ca ordinea de parcursere să fie de la adrese mici spre adrese mari. În acest caz, adresa primului element este adresa cea mai mică din sir.
- DF = 1 impune ca ordinea de parcursere să fie de la adrese mari spre adrese mici. În acest caz, adresa primului element este adresa cea mai mare din sir.

Pozitionarea prealabilă a lui DF se face folosind instrucțiunile **CLD** sau **STD**. Instrucțiunile care folosesc două siruri presupun că ambele siruri sunt parcuse în aceeași direcție.

d) Numărul de elemente, atunci când se dorește a fi exploatat, trebuie trecut în registrul CX.

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 Instrucțiuni pe șiruri pentru transferul de date

Aceste instrucțiuni sunt asemănătoare instrucțiunii **MOV**, dar ele realizează mai mult și operează mai rapid.

Instrucțiunea **LODS** se prezintă sub două forme: **LODSB** și **LODSW**. Ea încarcă un octet sau respectiv un cuvânt din memorie în registrul acumulator.

Tabelul 5. Instrucțiuni pe șiruri pentru transferul de date (String-Anweisungen für die Datenübertragung)

LODSB	$AL \leftarrow <DS:SI>$	if DF = 0 inc(SI) else dec(SI)
LODSW	$AX \leftarrow <DS:SI>$	if DF = 0 SI \leftarrow SI + 2 else SI \leftarrow SI - 2
STOSB	$<ES:DI> \leftarrow AL$	if DF = 0 inc(DI) else dec(DI)
STOSW	$<ES:DI> \leftarrow AX$	if DF = 0 DI \leftarrow DI + 2 else DI \leftarrow DI - 2
MOVSB	$<ES:DI> \leftarrow <DS:SI>;$	if DF = 0 {inc(ST); inc(DT)} else {dec(SI); dec(DI)}
MOVSW	$<ES:DI> \leftarrow <DS:SI>;$	if DF = 0 {SI \leftarrow SI + 2; DI \leftarrow DI + 2} else {SI \leftarrow SI - 2; DI \leftarrow DI - 2}

Instrucțiunea **LODSB** încarcă octetul de adresă DS:ESI în AL și 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 instrucțiunea **CLD**) atunci ESI este incrementat, iar dacă DF=1 (setarea acestuia cu 1 făcându-se cu instrucțiunea **STD**) ESI este decrementat. Această regulă impusă de valoarea din DF este valabilă pentru toate instrucțiunile pe șiruri ce afectează registri pointer. De exemplu:

va încărca AL cu conținutul octetului de deplasament 0 din cadrul segmentului de date și apoi se va incrementa ESI cu 1, acțiuni echivalente cu

4.2 String-Anweisungen für die Datenübertragung

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.

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:

```
cl  
mov SI,0  
lodsb
```

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
```

Instrucțiunea **LODSB** este însă mai rapidă și cu 2 octeți mai scurtă decât echivalentul

Der **LODSB**-Anweisung ist jedoch schneller und 2 Byte kürzer als der entsprechende

```
mov AL, [ESI]  
inc ESI
```

Instrucțiunea **LODSW** încarcă în AX cuvântul adresat de DS:ESI, incrementând sau decrementând apoi ESI cu 2 (deoarece este vorba de valori reprezentate pe cuvânt).

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).

Instrucțiunea **STOS** este complementara instrucțiunii **LODS**, ea transferând valoarea octet sau cuvânt din acumulator la locația de memorie de adresă ES:DI, incrementând sau decrementând apoi corespunzător pe DI. Și instrucțiunea **STOS** se prezintă sub două forme: **STOSB** și **STOSW**.

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**.

Instrucțiunea **STOSB** copiază octetul din AL la octetul de adresă ES:EDI, incrementând sau decrementând EDI în funcție de valoarea din DF.

Die **STOSB**-Anweisung kopiert das Byte von AL in das Adressbyte ES:EDI und erhöht oder verringert EDI abhängig vom Wert in DF.

Instrucțiunea **STOSW** este asemănătoare, copiind valoarea cuvântului din AX în cuvântul adresat de ES:EDI, incrementând sau decrementând apoi EDI cu 2.

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.

Instrucțiunile **LODS** și **STOS** funcționează eficient împreună pentru copierea de șiruri. Subrutina COPIERE de mai jos realizează copierea șirului terminat cu 0 care începe la DS:ESI în șirul ce începe la adresa ES:EDI:

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:

```
; Subrutină pentru copierea unui șir  
; terminat cu 0 în altul  
; Intrări: adresa de început a șirului sursă  
; DS:SI  
; adresa de început a șirului destinație ES:DI  
; Ieșiri: -  
; Regiștri afectați: AL, SI, DI
```

; 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 ;parcurgerea se va face crescător
; deci se va impune incrementarea

Copiere PROC

cld ; Die Reise wird inkrementell
; durchgeführt, sodass eine Erhöhung

iar: **lodsb** ; preia caracterul sursă
stosb ; memorează la destinație
cmp AL,0; a fost 0?
jnz iar ; dacă nu, se continuă
ret ; dacă da, se revine din
procedură

; 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

O modalitate mai eficientă de transfer a unui octet sau cuvânt dintr-o locație de memorie în alta este prin folosirea instrucțiunii **MOVS**. Aceasta este o combinație a instrucțiunilor **LODS** și **STOS**, ea preluând octetul (**MOVSB**) sau cuvântul (**MOVSW**) de la DS:ESI și 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

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 Instrucțiuni pe șiruri pentru consultarea și compararea datelor

Instrucțiunea **SCAS** (care are și ea două variante: **SCASB** și **SCASW**) caută în memorie o anumită valoare particulară (octet sau respectiv cuvânt).

Instrucțiunea **SCASB** compară valoarea din AL cu valoarea octet adresată de ES:EDI, setând *flag*-urile în acord cu rezultatele comparării (la fel ca și o instrucțiune **CMP**). După aceea, EDI este incrementat sau decrementat.

4.3 String-Anweisungen zum Abfragen und Vergleichen von Daten

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 șiruri pentru consultarea și 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 \leftarrow SI + 2; DI \leftarrow DI + 2 } ; else { SK-SI-2; DK-DI-2 }	OF, SF, ZF ,AF, PF, CF
--------------	---	-------------------------------

Instrucțiunile pe șiruri NU setează *flag*-urile în urma acțiunii asupra regiștrilor ESI, EDI sau ECX. Instrucțiunile **LODS**, **STOS** și **MOVS** nu afectează nici un *flag*, iar **SCAS** și **CMPS** modifică *flag*-urile doar ca rezultat al comparațiilor pe care le efectuează.

Instrucțiunea **SCASW** compară conținutul registrului AX cu cuvântul de adresă ES:EDI incrementând sau decrementând EDI cu 2, în funcție de valoarea din DF. Secvența de mai jos utilizează instrucțiunea **SCASW** cu prefixul **REPE** pentru a căuta ultima valoare nenulă dintr-un vector de întregi:

```

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      ; punе 0 в AX pentru a căuta un element nenul
; setze 0 in AX, um nach einem Nicht-Null-Element zu suchen
std             ; căutarea începe de la sfârșit
; 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 erschöpf sind
jne Gasit     ; dacă se ajunge aici, atunci tabloul conține 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

```

Instrucțiunea **CMPS** are ca rol efectuarea de comparații de șiruri de octeți sau cuvinte. Execuția unei instrucțiuni **CMPS** are ca efect comparaarea locațiilor de memorie de adrese DS:ESI și respectiv ES:EDI, urmată

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 durchgeföhrten 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:

Die **CMPS**-Anweisung hat die Aufgabe, Byte- oder Wortkettenvergleiche durchzuföhren. Die Ausführung eines **CMPS**-Anweisung hat den Effekt, die Speicherstellen von DS:ESI- und ES:EDI-

de incrementarea sau decrementarea registrelor SI și DI. *Flag-urile* vor fi actualizate corespunzător pentru a reflecta rezultatul comparării.

Instrucțiunea **CMPSB** realizează compararea la nivel de octet, iar **CMPSW** la nivel de cuvânt, prima incrementând sau decrementând ESI și EDI cu 1 iar ultima cu 2. În exemplul următor se compară două tablouri care conțin elemente reprezentate pe cuvânt pentru a se decide dacă primele 100 de elemente sunt sau nu identice:

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 regiștrii SI și 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 Execuția repetată a unei instrucțiuni pe șiruri

Pentru execuția repetată a unei instrucțiuni pe șiruri, limbajul de asamblare dispune de variante echivalente instrucțiunii de ciclare **LOOP**. Acestea sunt oferite de-așa numitele *prefixe de instrucțiune*. Sintaxa utilizării lor este:

4.4 Wiederholte Ausführung eines String-Anweisung

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 instrucțiune instrucțiune pe sir
(Anweisungspräfixe String-Anweisung)

În principiu este vorba despre un singur prefix de instrucțiune, și anume **REP**. În cazul utilizării instrucțiunii **SCAS** sau **CMPS** apar două variante posibile de **REP**, Prefixul de instrucțiune **REP** impune execuția repetată a instrucțiunii pe care o prefixează, până când valoarea din ECX devine 0. Dacă de la început avem ECX = 0, atunci instrucțiunea respectivă este inoperantă. Astfel, ciclul

este echivalent cu instrucțiunea

rep instrucțiune_pe_sir

În cazul utilizării prefixelor cu instrucțiunile **SCAS** sau **CMPS** condiția verificată se completează, ținându-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** – **REPeat while Equal** și **REPZ** – **REPeat while Zero**) provoacă execuția repetată a instrucțiunilor **SCAS** sau **CMPS** până când CX devine 0 sau până când apare o nepotrivire (caz în care ZF va primi valoarea 0).

Asemănător, **REPNE** (**REPeat while Not Equal**, formă echivalentă cu **REPNZ** – **REPeat if not Zero**) provoacă execuția repetată a instrucțiunii **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 instrucțiunilor **SCAS** și **CMPS** deoarece ele sunt singurele care afectează vreun *flag* (și anume ZF), permitând rafinarea condițiilor pe baza valorii *flag*-ului afectat. Rezultă deci, că în cazul instrucțiunilor **LODS**, **STOS** și **MOVS** toate cele cinci mnemonici prezентate au același efect:

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: *instrucțiune_pe_sir* (String-Anweisung)

loop Bucla

ist gleichbedeutend mit der Anweisung
(String-Anweisung)

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** – **REPeat while Equal** und **REPZ** – **REPeat while Zero**) 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** (**REPeat while Not Equal**, eine Form, die **REPNZ** – **REPeat if not Zero** 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

repetarea cât timp $CX \neq 0$.

Față de variantele cu **LOOP**, avantajul evident al folosirii acestor prefixe este comprimarea scrierii.

Mnemoniken den gleichen Effekt haben:
Wiederholung solange $CX \neq 0$.

Im Vergleich zu **LOOP**-Varianten liegt der offensichtliche Vorteil der Verwendung dieser Präfixe in der Schreibkomprimierung.

Curs 9

Programare multimodul în limbaj de asamblare

Contents

1. Programare modulară.....	2
1. Modular Programmierung.....	2
1.1 Cum se poate împărți o problemă în sub-probleme?.....	2
1.1 Wie können Sie ein Problem in Teilprobleme unterteilen?	2
1.2 Tehnici și instrumente	3
1.2 Techniken und Werkzeuge.....	3
1.2.1 Includerea statică la compilare/asamblare: directiva %include.....	3
1.2.1 Statische Einbeziehung bei der Kompilierung / Assemblierung: die Direktive %include.....	3
1.2.2 Legarea statică la linkeditare.....	7
1.2.2 Statische Bindung zu verlinkten.....	7
1.2.3 Cerințele NASM.....	11
1.2.3 Cerințele NASM.....	11
1.2.4 Folosirea în practică a directivelor global și extern.....	11
1.2.4 Anwendung von die Direktiven global und extern in der Praxis	11
1.2.5 Pașii necesari construirii programului executabil final	13
1.2.5 Die zum Erstellen des endgültigen ausführbaren Programms erforderlichen Schritte.....	13

Prezentăm forma generală a unui program în NASM, însățită de un scurt exemplu:

1. Programare modulară

1.1 Cum se poate împărți o problemă în sub-probleme?

Modularizarea este un concept care are sensuri diferite în funcție de nivelul de abstractizare la care este aplicat:

- La nivelul programelor, modularizarea constă în crearea de unități logice;
- La nivelul codului sursă (al unităților), modularizarea constă în crearea de fișiere distincte;
- La nivelul fișierelor, modularizarea constă în crearea de subrute.

Wir präsentieren die allgemeine Form eines Programms in NASM, begleitet von einem kurzen Beispiel:

1. Modular Programmierung

1.1 Wie können Sie ein Problem in Teilprobleme unterteilen?

Modularisierung ist ein Konzept, das je nach Abstraktionsebene, auf die es angewendet wird, unterschiedliche Bedeutungen hat:

- Auf Programmebene besteht die Modularisierung in der Erstellung logischer Einheiten;
- Auf der Quellcode-Ebene (der Einheiten) besteht die Modularisierung darin, separate Dateien zu erstellen;
- Auf Dateiebene besteht die Modularisierung aus der Erstellung von Unterprogrammen.

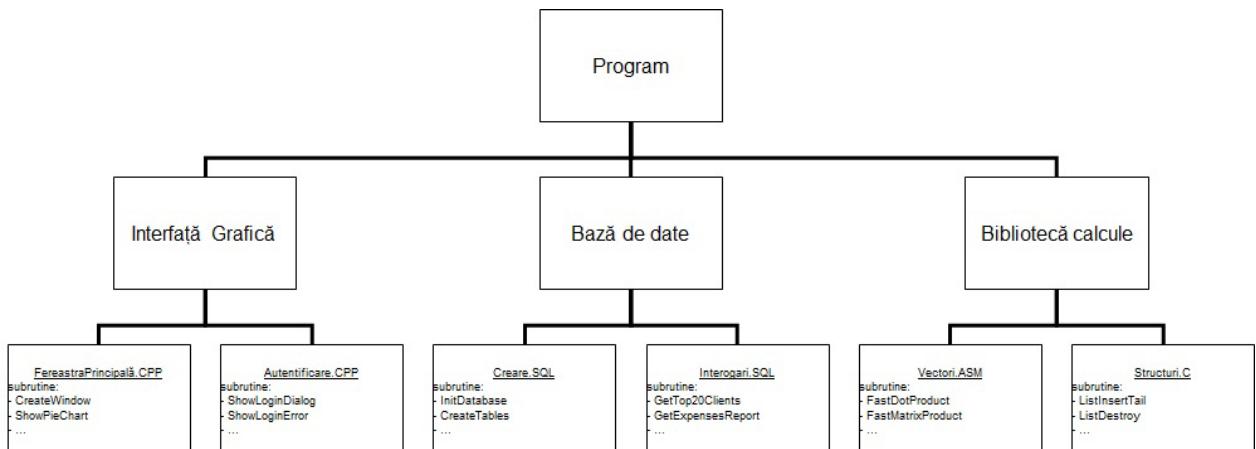


Figura 1. Conceptul de modularizare, la diferite niveluri de abstractizare (Das Konzept der Modularisierung auf verschiedenen Abstraktionsebenen)

În momentul creării unei aplicații mai complexe, trebuie să ne întrebăm care sunt sub-problemele pentru care există deja rezolvări disponibile, astfel încât să nu rescriem cod în mod nенесар.

Bei der Erstellung einer komplexeren Anwendung müssen wir uns fragen, für welche Unterprobleme bereits Lösungen verfügbar sind, damit wir den Code nicht unnötig umschreiben.

Reutilizarea codului poate să se refere la următoarele aspecte:

- La nivelul fișierelor sursă, putem refolosi
 - Cod sursă și date din asamblare
 - Directiva **%include**
- La nivelul fișierelor binare, putem refolosi
 - Cod sursă și date din asamblare
 - Cod și date din limbaje de nivel înalt
 - Biblioteci

Observație: termenul „a refolosi” nu trebuie înțeles în sensul reutilizării unor porțiuni de cod scrise de noi, ci a reutilizării unor porțiuni de cod în general (chiar și scrise de altcineva).

1.2 Tehnici și instrumente

1.2.1 Includerea statică la compilare/asamblare: directiva **%include**

Această directivă este specifică limbajului de asamblare, dar are echivalent și în alte limbiage.

Este foarte important de subliniat faptul că modularizarea permite doar *divizarea codului* scris în limbajul respectiv (nu se pot combina coduri sursă scrise în mai multe limbiage).

Prin definiție, fiecare modul este o unitate de sine stătătoare și grupează mai multe funcții și variabile înrudite.

Principalele avantaje ale programării modulare sunt:

- Abstractizare: de exemplu, în C fiecare modul conține o interfață (un *header*, extensia .h, .hpp sau fără nici o extensie cum e cazul header-ului din biblioteca standard cpp) și o implementare (fișierul sursă). În

Die Wiederverwendung des Codes kann sich auf folgende Aspekte beziehen:

- Auf Quelldateiebene können wir wiederverwenden
 - Quellcode und Assemblierungsdaten
 - Die Direktive **%include**
- Auf der Ebene der Binärdateien können wir wiederverwenden
 - Quellcode und Assemblierungsdaten
 - Code und Daten auf Hochsprachen
 - Bibliotheken

Hinweis: Der Begriff „Wiederverwendung“ ist nicht in dem Sinne zu verstehen, dass einige von uns geschriebene Codeteile wiederverwendet werden, sondern dass einige Codeteile im Allgemeinen wiederverwendet werden (selbst wenn sie von jemand anderem geschrieben wurden).

1.2 Techniken und Werkzeuge

1.2.1 Statische Einbeziehung bei der Kompilierung / Assemblierung: die Direktive **%include**

Diese Direktive ist spezifisch für die Assemblersprache, hat jedoch eine Entsprechung in anderen Sprachen.

Es ist sehr wichtig zu betonen, dass die Modularisierung nur die *Aufteilung des geschriebenen Codes* in die jeweilige Sprache ermöglicht (es ist nicht möglich, in mehreren Sprachen geschriebene Quellcodes zu kombinieren).

Per Definition ist jedes Modul eine eigenständige Einheit und gruppiert mehrere Funktionen und zugehörige Variablen.

Die Hauptvorteile der modularen Programmierung sind:

- Abstraktion: Beispielsweise enthält in C jedes Modul eine Schnittstelle (ein *header*, eine Erweiterung mit der Endung .h, .hpp oder ohne eine Erweiterung wie die *header* der CPP-Bibliothek) und eine

interfață doar se *declară* funcțiile, iar *definirea* lor propriu zisă se realizează în fișierul sursă. Astfel, se asigură abstractizarea: se pune accentul pe elementele esențiale (CE face modulul) și se ignoră detaliile (CUM sunt implementate aceste funcții).

- Prin această *ascundere a implementării* în fișierul sursă se asigură *corectitudinea* modulului (utilizatorul nu poate să modifice codul sursă) și *transparența* (se pot face modificări ulterioare asupra codului sursă și utilizatorul să nu fie conștient de acestea dacă nu se modifică interfața funcțiilor). În plus, dacă eventual codul din implementare este furnizat în mod binar (de exemplu, o bibliotecă statică, fișier obiect etc.), utilizatorii (sau eventuali atacatori ai aplicației) nu pot face *reverse engineering* ca să vadă codul sursă. În plus, utilizatorul modulului poate să utilizeze doar funcțiile expuse în interfață.

Totuși, utilizarea directivei **include** nu este programare multimodul autentică! **DE CE?** Această metodă implică anumite riscuri importante.

1. Prin utilizarea ei se expune codul sursă!

Obținerea unui fișier executabil trece prin 3 etape principale: preprocesare, compilare (asamblare) și linkeditare.

Implementierung (Quelldatei). In der Schnittstelle werden nur die Funktionen *deklariert* und in der Quelldatei eine eigene *Definition* realisiert. Somit ist die Abstraktion gewährleistet: Es wird Wert auf das Wesentliche gelegt (WAS stellt das Modul) und Details werden ignoriert (WIE diese Funktionen implementiert werden).

- Diese *Verschleierung der Implementierung* in der Quelldatei stellt die *Korrektheit* des Moduls (der Benutzer kann den Quellcode nicht ändern) und die *Transparenz* sicher (spätere Änderungen am Quellcode können vorgenommen werden, und der Benutzer ist sich dessen nicht bewusst, wenn die Funktionsschnittstelle nicht geändert wird). Wenn der Code in der Implementierung in binärer Form bereitgestellt wird (z. B. eine statische Bibliothek, eine Objektdaten usw.), können Benutzer (oder mögliche Angreifer der Anwendung) den Quellcode nicht rückentwickeln (*reverse engineering*). Außerdem kann der Modulbenutzer nur die Funktionen verwenden, die in der Schnittstelle verfügbar sind.

Die Verwendung der Direktive **include** jedoch keine authentische Multimodulprogrammierung! **WARUM?** Diese Methode birgt bestimmte wichtige Risiken.

1. Durch die Verwendung wird der Quellcode verfügbar gemacht!

Das Abrufen einer ausführbaren Datei erfolgt in drei Hauptschritten: Vorverarbeitung, Kompilieren (Assemblierung) und Verknüpfen (linkediting).

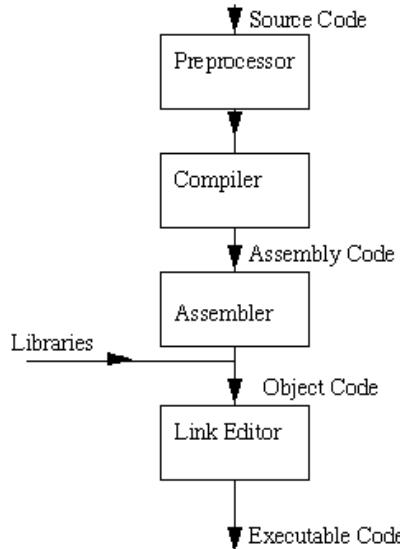


Figura 2. Fazele obținerii unui fișier executabil (Die Schritte zum Abrufen einer ausführbaren Datei)

Mecanismul de preprocessare implică o concatenare textuală a fișierelor, astfel că se poate obține un fișier sursă mult mai mare decât cel inițial (care poate fi relativ scurt, dar care poate conține multe „**include**”-uri). După etapa de preprocessare, un atacator are deja acces la tot codul sursă! În schimb, dacă am împărți codul sursă în module care să fie asamblate separat, putem „transporta” cu noi doar fișierele .obj, pe care doar noi știm cum se leagă de fișierul principal (deci putem de pildă transporta mai multe fișiere .obj, dintre care nu toate sunt utile aplicației noastre). Este mult mai greu de făcut *reverse engineering* pe un fișier .obj decât pe un fișier sursă.

2. Expune cu vizibilitate globală toate denumirile de variabile, constante etc. Astfel apar conflicte la redefiniții / redeclarări. De exemplu, dacă avem două variabile cu același nume declarate în două module diferite, dacă *includem* aceste variabile în același fișier vom avea un conflict de nume. În schimb, dacă folosim variabile într-un context multi-modul, atunci nu va fi nici o

Vorverarbeitungsmechanismus beinhaltet die Verkettung von Dateien in Textform, sodass eine viel größere Quelldatei als die ursprüngliche Datei erhalten werden kann (die relativ kurz sein kann, aber viele „**include**“ enthalten kann). Ein Angreifer hat nach der Vorverarbeitung bereits Zugriff auf den gesamten Quellcode! Wenn wir stattdessen den Quellcode in Module aufteilen, die separat assembleieren werden sollen, können wir nur die .obj Dateien „mitnehmen“, die wir nur mit der Hauptdatei verknüpfen können (sodass wir beispielsweise mehr .obj-Dateien führen können, von denen nicht alle für unsere Anwendung nützlich sind). *Reverse Engineering* ist für eine .obj-Datei viel schwieriger als für eine Quelldatei.

2. Es macht mit globaler Sichtbarkeit alle Namen von Variablen, Konstanten usw. verfügbar. Somit ergeben sich Konflikte in den Redefinitionen / Redeklarationen. Wenn beispielsweise zwei Variablen mit demselben Namen in zwei verschiedenen Modulen deklariert sind und diese Variablen in derselben Datei *enthalten* sind, liegt ein Namenskonflikt vor. Im Gegensatz dazu gibt

problemă.

Prin directiva **include**, includem fișierul în întregime – și ce este util, și ce nu!

es kein Problem, wenn wir Variablen in einem Kontext mit mehreren Modulen verwenden.

Durch die **include**-Direktive wird die gesamte Datei eingeschlossen – und was ist nützlich und was nicht!

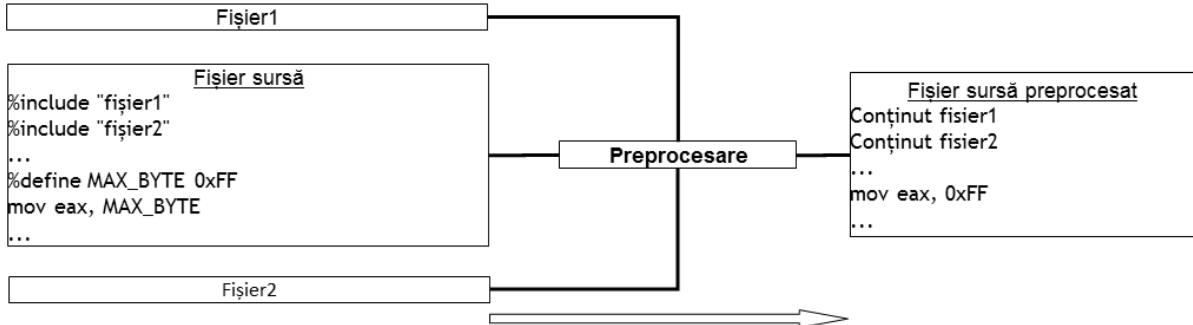


Figura 3. Obținerea fișierului executabil: faza de preprocesare (Erhalten der ausführbaren Datei: Vorverarbeitungsphase)

1) Exemplu de folosire **%include**

; fișierul constante.inc
; gardă dublă-includere

```

%ifndef _CONSTANTE_INC_      ; la prima includere, _CONSTANTE_INC_ nu este definit
                                ; Beim ersten Einschluss ist _CONSTANTE_INC_ nicht definiert
#define _CONSTANTE_INC_ ; definim _CONSTANTE_INC_ -> condiție falsă la viitoare includeri
                                ; wir definieren _CONSTANTE_INC_ -> falsche Bedingung für zukünftige Aufnahme

; se recomandă ca astfel de fișiere (incluse de către altele) să conțină (doar) declarații!
; Es wird empfohlen, dass solche Dateien (von anderen eingeschlossen) (nur) Deklarationen enthalten!

```

MAX_BYTE	equ	0xFF
MAX_WORD	equ	0xFFFF
MAX_DWORD	equ	0xFFFFFFFF
MAX_QWORD	equ	0xFFFFFFFFFFFFFFFF

%endif; _CONSTANTE_INC_

2) Exemplu folosire **%include** – împachetare EAX într-un BYTE / WORD / DWORD, conform magnitudinii valorii acestuia
; fișierul program.asm

%include "constante.inc"

1) Anwendungsbeispiel **%include**

; fișierul constante.inc
; gardă dublă-includere

2) Anwendungsbeispiel **%include** - EAX-Packung in einem BYTE / WORD / DWORD, abhängig von der Größe seines Wertes

```

cmp EAX, MAX_BYTE           ; începe valoarea din EAX într-un byte?
ja .nu_incape_in_octet      ; Passt der Wert in EAX in ein Byte?

.incape_in_octet:
mov [rezultat_octet], AL     ; dacă da, salvăm AL în rezultat_octet
jmp .gata                   ; Wenn ja, speichern wir AL in rezultat_octet

.nu_incape_in_octet:
cmp EAX, MAX_WORD          ; altfel verificăm dacă este suficient un WORD
ja .nu_incape_in_cuvant    ; ansonsten prüfen wir, ob ein WORD ausreicht

.incape_in_cuvant:
mov [rezultat_word], AX      ; dacă da, salvăm AX în rezultat_word
jmp .gata                   ; Wenn ja, speichern wir AX in rezultat_word

.nu_incape_in_cuvant:
mov [rezultat_dword], EAX    ; dacă nu este suficient un WORD, salvăm întregul EAX
.gata:                         ; Wenn ein WORT nicht ausreicht, speichern wir den gesamten EAX

```

1.2.2 Legarea statică la linkeditare

Acesta este un pas realizat de către un linkeditor după asamblare/compilare.

1.2.2 Statische Bindung zu verlinkten

Dies ist ein Schritt, den ein Linker nach der Assemblierung / Kompilierung ausführt.

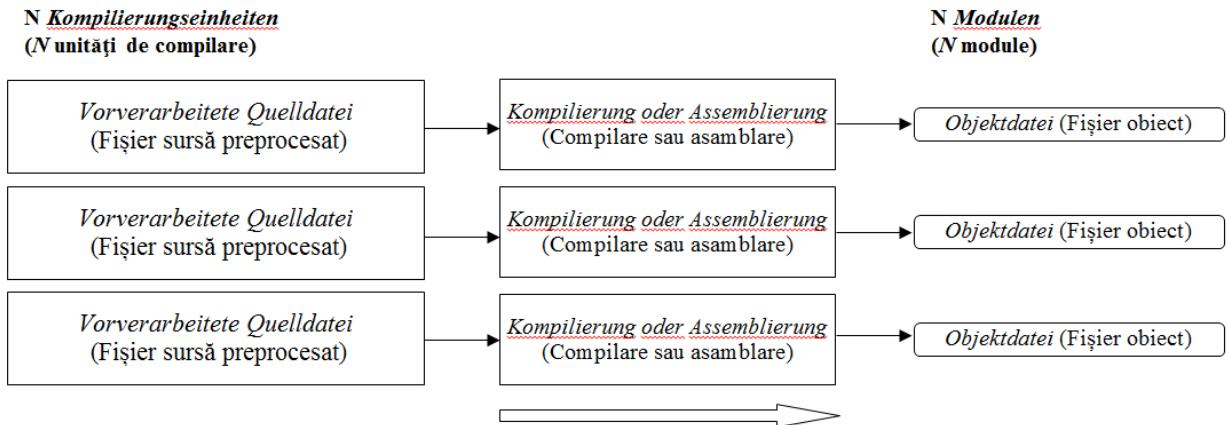


Figura 4. Obținerea fișierului executabil: faza de compilare sau asamblare (Erhalten der ausführbaren Datei: Kompilierungs- oder Assemblierungsphase)

N Modulen (N module)

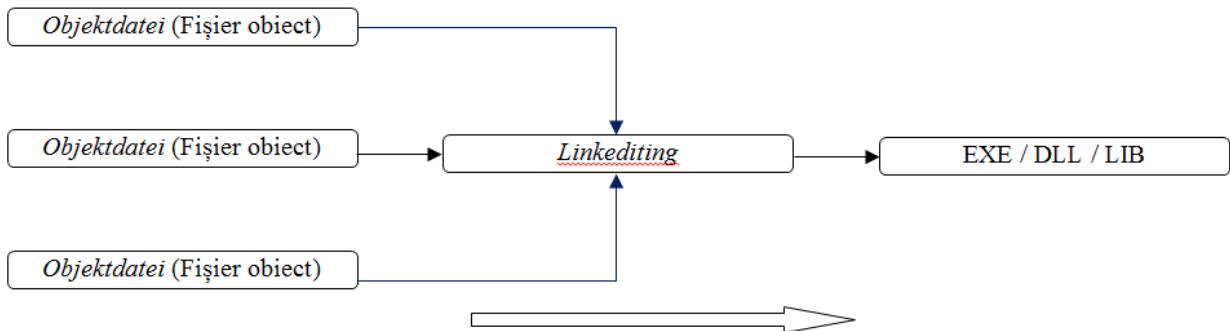


Figura 5. Obținerea fișierului executabil: faza de linkeditare (Erhalten der ausführbaren Datei: Verlinktensphase)

Preprocesorul: transformă text în text.

Efectuează prelucrări asupra textului sursă, rezultând un text sursă intermediu. Se poate imagina ca fiind o componentă a compilatorului sau asamblorului.

Poate lipsi, multe limbaje nu au un preprocesor!

Asamblorul: transformă instrucțiunile (text) în codificare binară (fișier obiect).

Codifică instrucțiunile și datele (variabilele) din textul sursă preprocesat și construiește un fișier obiect care conține cod mașină și valori de variabile, alături de informații despre conținut (denumiri de variabile, subroutines, informații despre tipul și vizibilitatea acestora etc.).

Compilatorul: transformă instrucțiunile (text) în codificare binară (fișier obiect)

Identifică secvențe de instrucțiuni de procesor prin care se pot obține funcționalitățile descrise în textul sursă iar apoi, precum un asamblor, generează un fișier obiect care conține codificarea binară a

Präprozessor: Konvertiert Text in Text.

Es führt eine Verarbeitung des Quelltextes durch, was zu einem Zwischenquelltext führt. Es kann als eine Komponente des Compilers oder Assemblers vorgestellt werden.

Es kann fehlen, viele Sprachen haben keinen Präprozessor!

Assemblierer (Assembler): wandelt Anweisungen (Text) in binäre Codierung (Objektdatei) um.

Es codiert Anweisungen und Daten (Variablen) im vorverarbeiteten Quelltext und erstellt eine Objektdatei, die Maschinencode und Variablenwerte sowie Inhaltsinformationen (Variablennamen, Unterprogramme, Informationen zu Typ und Sichtbarkeit usw.) enthält.

Compiler: wandelt Anweisungen (Text) in binäre Codierung (Objektdatei) um.

Identifiziert Sequenzen von Prozessoranweisungen, über die die im Quelltext beschriebenen Funktionalitäten abgerufen werden können, und generiert dann als Assembler eine Objektdatei, die

acestora și a variabilelor din program.

Asamblarea este un caz special de compilare, unde instrucțiunile de procesor sunt gata oferite direct în textul programului și ca atare nu necesită să fie alese de către compilator!

Linkeditor: fișiere obiect => bibliotecă sau program

Construiește rezultatul final, adică un program (.exe) sau bibliotecă (.dll sau .lib) în care leagă împreună (include) codul și datele binare prezente în fișierele obiect.

Nu contează ce compilatoare sau ce limbi de programare au fost folosite! Legarea necesită doar ca fișierele de intrare să respecte formatul standard al fișierelor obiect!

Legarea statică la linkeditare permite unirea mai multor module binare (fișiere obiect sau biblioteci statice) într-un singur fișier.

La intrări putem avea oricâte fișiere obiect (.OBJ) și / sau biblioteci statice (.LIB).

Atenție, nu toate fișierele .LIB sunt biblioteci statice!

La iesiri se obțin fișiere .EXE sau .LIB sau .DLL (Dynamic-Link Library).

Programarea multimodul înseamnă că oricâte fișiere pot fi compilate separat și linkeditate împreună. Este un pas realizat de către linkeditor, după compilare / asamblare, deci nu depinde de limbaj!

ihre binäre Codierung und Programmvariablen enthält.

Assemblierung ist ein spezieller Kompilierungsfall, bei dem Prozessoranweisungen direkt im Programmtext bereitgestellt werden und als solche nicht vom Compiler ausgewählt werden müssen!

Linkeditor: Objektdateien => Bibliothek oder Programm

Erstellt das Endergebnis, deshalb ein Programm (.exe) oder eine Bibliothek (.dll oder .lib), die den in den Objektdateien vorhandenen Binärkode und die darin enthaltenen Daten verknüpft (enthält).

Es ist egal, welche Compiler oder Programmiersprachen verwendet wurden! Das Binden setzt nur voraus, dass die Eingabedateien dem Standardformat der Objektdateien entsprechen!

Durch die statische Bindung an einen Link können Sie mehrere Binärmodule (Objektdateien oder statische Bibliotheken) in einer einzigen Datei zusammenführen.

Bei Einträgen können wir beliebige Objektdateien (.OBJ) und / oder statische Bibliotheken (.LIB) haben.

Achtung, nicht alle .LIB-Dateien sind statische Bibliotheken!

Die Ausgaben erhalten .EXE- oder .LIB- oder .DLL-Dateien (Dynamic Link Library).

Multimodale Programmierung bedeutet, dass jede Datei separat kompiliert und miteinander verknüpft werden kann. Dies ist ein Schritt, den der Link nach dem Kompilierung / Assemblierung ausführt, es kommt also nicht auf die Sprache an!

Reutilizarea codului:

1. În formatul binar nu este expus codul sursă.
2. Permite inter-operabilitate între limbi diferite!

Alte avantaje și dezavantaje:

1. Editorul de legături poate identifica și elimina resurse neutilizate sau efectua alte optimizări;
2. Dimensiune mare a programului: programul înglobează resursele externe reutilizate;
3. Dimensiune mare a programelor: bibliotecile populare duplicate în multe programe.

În NASM avem directivele **global** (mecanism export) și **extern** (mecanism import)

- **global nume** – oferă posibilitatea de utilizare din exterior a resursei respective, specificate prin nume
- **extern nume** – solicitare de acces la resursa specificată; necesită să fie publică!

Wiederverwendung von Code:

1. Der Quellcode wird nicht im Binärformat angezeigt.
2. Ermöglicht die Interoperabilität zwischen verschiedenen Sprachen!

Andere Vor- und Nachteile:

1. Der Link-Editor kann nicht verwendete Ressourcen identifizieren und beseitigen oder andere Optimierungen durchführen.
2. Große Programmgröße: Das Programm enthält wiederverwendete externe Ressourcen
3. Große Programmgröße: Beliebte Bibliotheken, die in vielen Programmen doppelt vorhanden sind.

In NASM gibt es **global** (Exportmechanismus) und **extern** (Importmechanismus) Direktiven

- **global Name** – bietet die Möglichkeit der externen Verwendung der jeweiligen Ressource, angegeben durch den Namen
- **extern Name** – Anforderung des Zugriffs auf die angegebene Ressource; muss öffentlich sein!

1.2.3 Cerințele NASM

Este foarte important de reținut că resursele sunt partajate de comun acord.

Exportul se realizează prin directiva **global** nume1, nume2, ... Prin aceasta resursele respective se pun la dispoziția oricărui fișier ar fi interesat.

Importul se realizează prin directiva **extern** nume1, nume2, ... Prin aceasta se solicită accesul la resursele respective, indiferent din ce fișier vor fi oferite.

Solicitare fără disponibilitate = eroare!

Nu se pot importa decât resurse care sunt exportate undeva. Însă disponibilitate fără solicitare este caz permis, deoarece chiar dacă nici un modul din program nu solicită / folosește resursele respective, poate ele vor fi utilizate într-o versiune viitoare sau de către un alt program.

Limbajele de programare de nivel mai înalt oferă și ele la rândul lor construcții sintactice cu rol echivalent (exemplu: în limbajul C, disponibilitatea este automată / implicită, putându-se însă opta pentru a bloca accesul prin folosirea cuvântului cheie **static**. Solicitarea de acces se face (tot) prin intermediu cuvântului cheie **extern**).

1.2.4 Folosirea în practică a directivelor global și extern

1.2.3 Cerințele NASM

Es ist sehr wichtig, sich daran zu erinnern, dass die Ressourcen in gegenseitigem Einvernehmen geteilt werden.

Der Export erfolgt über die Direktive **global** name1, name2, ... Hierdurch werden die jeweiligen Ressourcen für jede interessierte Datei zur Verfügung gestellt.

Der Import erfolgt über die Direktive **extern** name1, name2, ... Damit wird der Zugriff auf die jeweiligen Ressourcen angefordert, unabhängig davon, welche Datei angeboten wird.

Anfrage ohne Verfügbarkeit = Fehler!

Es können nur exportierte Ressourcen importiert werden. Eine Verfügbarkeit ohne Anfrage ist jedoch zulässig, da selbst wenn kein Modul im Programm die entsprechenden Ressourcen anfordert / verwendet, diese möglicherweise in einer zukünftigen Version oder von einem anderen Programm verwendet werden.

Höhere Programmiersprachen bieten auch syntaktische Konstruktionen mit gleicher Funktion (z. B. in der Sprache C ist die Verfügbarkeit automatisch / standardmäßig, Sie können den Zugriff jedoch mithilfe des Schlüsselworts **static** blockieren. Die Anforderung für den Zugriff lautet mache (alles) durch das **extern** Schlüsselwort).

1.2.4 Anwendung von die Direktiven global und extern in der Praxis

; FIŞIER1.ASM

global Var1, Subrutina2

extern Var3, Subrutina3

Subrutina1:

....

apel(Subrutina3)

....

operatii(Var3)

....

Subrutina2:

....

Var1 dd ...

Var2 db ...

; FIŞIER1.ASM

global Var1, Subrutina2

extern Var3, Subrutina3

Subrutina1:

....

apel(Subrutina3)

....

operatii(Var3)

....

Subrutina2:

....

Var1 dd ...

Var2 db ...

Exemplu de program multimodul NASM +
NASM

; FIŞIER2.ASM

extern Var1, Subrutina2

global Subrutina3, Var3

Subrutina3:

....

apel(Subrutina2)

....

operatii(Var1)

....

Subrutina1:

....

Var2 db ...

Var3 dd ...

; FIŞIER2.ASM

extern Var1, Subrutina2

global Subrutina3, Var3

Subrutina3:

....

apel(Subrutina2)

....

operatii(Var1)

....

Subrutina1:

....

Var2 db ...

Var3 dd ...

Beispiel eines multimodul Programms
NASM + NASM

Pot fi refolosite
denumirile cât timp
nu sunt globale!

```

; MODULUL MAIN.ASM                                ; MODULUL SUB.ASM
global SirFinal          →      extern SirFinal
extern Concatenare       ←      global Concatenare

import printf msvcrt.dll
import exit msvcrt.dll
extern printf, exit
global start

segment code use32 public code class='code'
start:
    mov eax, Sir1
    mov ebx, Sir2
    call Concatenare
    push dword SirFinal
    call [printf]
    add esp, 1*4
    push dword 0
    call [exit]

segment data use32
Sir1 db 'Buna ', 0
Sir2 db 'dimineata!', 0
SirFinal resb 1000 ; spatiu pentru rezultat

; MODULUL SUB.ASM
; eax = adresa primului sir, ebx = adresa sirului secund
Concatenare:
    mov edi, SirFinal ; destinatie = SirFinal
    mov esi, eax      ; sursa = primul sir
.sir1Loop:
    lodsb             ; luam octetul urmator
    test al, al        ; este terminatorul de sir (=0)?
    jz .sir2           ; daca da, trecem la sirul al doilea
    stosb             ; (altfel) copiem in destinatie
    jmp .sir1Loop     ; si continuam pana la nul
.sir2:
    mov esi, ebx      ; sursa = sirul al doilea
.sir2Loop:
    lodsb             ; acelasi proces pentru noul sir
    test al, al
    jz .gata
    stosb
    jmp .sir2Loop
.gata:
    stosb             ; adaugam terminatorul de sir din al
ret

```

1.2.5 Pașii necesari construirii programului executabil final

1. Se asamblează fișierul main.asm

nasm.exe -fobj main.asm

2. Se asamblează fișierul sub.asm

nasm.exe -fobj sub.asm

3. Se editează legăturile dintre cele două module

alink.exe main.obj sub.obj -oPE -entry:start -subsys:console

Observații: cele două module pot fi asamblate în orice ordine! Abia în timpul linkeditării este necesar ca simbolurile referite să aibă, toate, implementare disponibilă în unul dintre fișierele obiect oferite linkeditorului.

Linkeditarea, în mod evident, este posibilă doar după asamblare / compilare!

1.2.5 Die zum Erstellen des endgültigen ausführbaren Programms erforderlichen Schritte

1. Assemblieren Sie die Datei main.asm

nasm.exe -fobj main.asm

2. Assemblieren Sie die Datei sub.asm

nasm.exe -fobj sub.asm

3. Die Verbindungen zwischen den beiden Modulen werden bearbeitet

Anmerkungen: Die beiden Module können in beliebiger Reihenfolge assemblieren werden! Nur während der Verlinkung ist es erforderlich, dass die referenzierten Symbole alle in einer der dem Verlinker angebotenen Objektdateien implementierbar sind.

Eine Verknüpfung ist natürlich nur nach Assemblierung / Kompilierung möglich!

Curs 10

Recapitulare

- 1) <https://nasm.us/doc/>
- 2) Macrouri v.s. Proceduri

Macrouri	Proceduri
<ul style="list-style-type: none">- la fiecare apel se copiază secvența de instrucțiuni- nu sunt necesare instrucțiuni de apel (CALL) și de revenire din rutină (RET)- nu se folosește stiva- transferul de parametri se face prin copierea numelui	<ul style="list-style-type: none">- o singură copie pentru mai multe apeluri- se folosesc instrucțiuni de apel (CALL) și de revenire (RET)- se utilizează stiva la apel și la revenire- transferul de parametri se face prin registre sau stivă

- 3) Problema etichetelor in macrouri

La copierea repetată, etichetele se duplicează. Soluția: folosirea etichetelor „locale”

Directiva LOCAL

```
<nume_macro> macro [<par1> [<par2> ...]]  
local <et1> [<et2> ..]]  
.....  
<et1>:  
.....  
<et2>:  
endm
```

4) Avantajele și dezavantajele utilizării macrourilor

Avantaje	Dezavantaje
1) pot fi create „instrucțiuni” noi 2) poate duce la o programare mai eficientă 3) execuție mai eficientă în comparație cu apelurile de proceduri	1) pot ascunde operații care afectează conținutul regisitrelor 2) utilizarea extensivă a macrourilor îngreunează înțelegerea și menținerea programului

5) Problema optimizării

Când și ce se optimizează

- regula 90/10 - 90% din timp se execută 10% din cod
 - consecință – dacă se elibera 90% din codul rar folosit îmbunătățirea este de 10%
 - ce se dorește?
- timp redus sau resurse ocupate (în special memorie) redusă
 - Cum se măsoară timpul ocupat al procesorului pentru fiecare modul – cu programe de tip „profiler”
 - când este bine să se optimizeze:
 - de la început:
 - se optimizează și partea nesemnificativă
 - programul se scrie greu și se înțelege și mai greu
 - la sfârșit:
 - prea târziu

Optimizarea este necesară?

Contra-argumente:

- viteza mare a procesoarelor, a memoriilor și a magistralelor
- spații de memorie foarte mari

Când este necesară optimizarea:

- prelucrări de informații multimedia
- prelucrări de semnale

- sisteme de control în timp real, sisteme reactive

Trei tipuri de optimizare

- optimizare de nivel înalt
Implicită găsirea unui alt algoritm, mai bun, de pildă cu grad mai mic de complexitate (de exemplu $O(n^2) \Rightarrow O(n \log(n))$)
- optimizare de nivel mediu
Algoritm implementat mai bine
- optimizare de nivel scăzut
Minimizarea numărului de cicluri

Tehnici de optimizare

- reducerea numărului de bucle imbricate
- reducerea timpului de execuție al buclei interioare
- utilizarea pointerilor în adresarea elementelor unor structuri de date
- parcurgerea structurilor de date prin incrementarea și decrementarea poantelor în locul calculării adresei elementului
–ex: $\text{tab}[i][j]$
 $\text{adr_element}_{ij} = \text{adresa}(\text{tab}) + i * \text{lung_rand} + j$
- utilizarea registrelor interne ale procesorului în operațiile curente

Exemplu de optimizare

Problema: filtrarea unei imagini de 100 de ori, rezoluție 256 * 256 pixeli

Variante:

- program Pascal - 45s
- program C - 29s
- asamblare (V1) - 6s
 - s-au folosit deplasamente precalculate în locul calculului de adresă prin indecsă
- asamblare (V2) - 4s
 - s-au folosit registre interne în locul variabilelor de memorie
- asamblare (V3) - 2,5s
 - s-a evitat recopierea imaginii intermediare în matricea inițială

- asamblare (V4) 2,4s
 - s-a redus dimensiunea variabilelor de la întreg la caracter pentru a beneficia de memoria cache
- asamblare (V5) 2,2s
 - s-a schimbat algoritmul de filtrare, s-au luat în calcul numai 4 vecini ai pixelului în loc de 8 (rezultatul este diferit)

6) Apelul procedurilor scrise în asamblare din limbaje de nivel înalt

Când se justifică:

- la accesul direct al unor resurse fizice (de exemplu: interfețe de I/E, memoria video etc.)
- pentru creșterea eficienței unor secvențe critice
- atunci când funcțiile de acces la resurse au fost scrise (deja) în asamblare

Dificultăți:

- altă filozofie de scriere a programelor (ex: registre în loc de variabile, date simple în locul celor structurate)
- transferul parametrilor de apel și a rezultatelor

Reguli:

- procedura scrisă în limbaj de asamblare se va declara “far” și “public”
- numele și tipul procedurii se va declara în limbajul de nivel înalt cu mențiunea EXTERN
- parametrii de apel se transmit prin stivă:
 - ordinea de scriere pe stivă coincide cu ordinea din lista de parametri
 - primul parametru este cel mai adânc în stivă (se află la o adresă mai mare)
 - ultima dată înregistrată pe stivă este un dublucuvânt care reprezintă adresa de revenire din rutină