4. tétel

a. Összeadás, kivonás és szorzás műveletek, memóriaszegmensek kezelése

Összeadás

A bináris összeadást hasonlóan végezzük, mint a decimális számoknál: a számokat összeadjuk az adott helyi értéken (**A**+**B**=**S**), és ha van maradék (átvitel) (**C**), akkor azt hozzáadjuk a magasabb helyi értékek összegéhez.

A		**B**		**S**	**C**
0	+	0	=	0	0
0	+	1	=	1	0
1	+	0	=	1	0
1	+	1	=	1	1

Az összeadás műveleti táblája nem más, mint az **A** és **B** értékekkel elvégzett XOR logikai művelet, illetve a maradék és a két érték AND kapcsolata. Így összeadó készíthető egy XOR és AND kapu-áramkörből. Mivel ez a megoldás nem számol a korábban keletkezett maradékkal, ezért ez csak fél-összeadó. Teljes összeadó készítéséhez két fél-összeadó áramkör kell.

Mivel a számítógépek nem egybites számokkal dolgoznak, ezért az azonos helyi értéken lévő biteket egymás után össze kell adni, és a keletkezett átvitelt hozzá kell adni a következő helyi értékek összegéhez. Összeadásnál figyelni kell a túlcsordulásra.

Kivonás

A számítógépekben a kivonáshoz is az összeadó áramköröket használják, de akkor a kivonandó kettes komplemensét veszik, és úgy végzik el a műveletet.
Példa: 11101110
Egyes komplemens: 0001 0001
Kettes komplemens: hozzáadunk 1-et: 0001 0010 (és ezt adjuk hozzá az eredeti számhoz, így a kivont értéket fogjuk kapni)

Szorzás

A bináris szorzást úgy végezzük, mintha decimális számok lennének: a szorzandót megszorozzuk egyesével a szorzó egyes helyi értékein álló számmal. A részszorzatokat úgy írjuk le, hogy mindig egy helyi értékkel jobbra toljuk azokat, majd az így kapott részeredményeket összeadjuk.

A		**B**		**S**
0	*	0	=	0
0	*	1	=	0
1	*	0	=	0
1	*	1	=	0

Az eredmény az **A** és a **B** értékkel elvégzett AND logikai művelet.

Bájtos szorzásnál legalább kétbájtnyi adatterületre van szükség az eredmény tárolásához, így az eredmény egy kétbájtos regiszterbe kerül. Ha kétbájtos adatokkal dolgozunk, akkor négybájtos lesz a szorzat.

b. A többmenetes fordítók működése

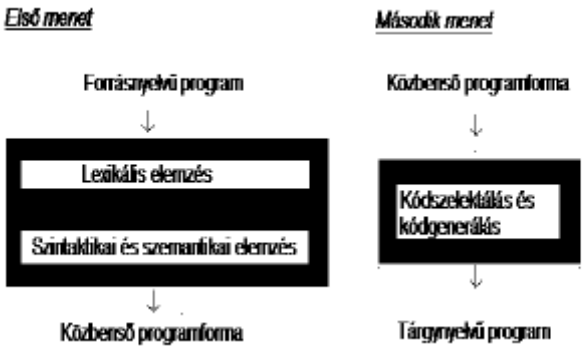
2.2.2. Többmenetes fordítás

Többmenetes fordításról beszélünk, ha a fázisok több különböző menetben futnak le. Ilyenkor szükség van közbenső programformák tárolására. Ezek az egyes menetek outputjai, és más menetek inputjai.

Fontos megjegyezni, hogy nem csak az fordulhat elő, hogy egy menetben több fázis fut le, hanem az is, hogy egy fázis több menetre van osztva. Akárhogy is, a fázisoknak olyan sorrendben kell egymást követniük, ahogy fent szerepelnek.

Néhány esetben az egyetlen lehetőség a többmenetes fordítás, mert a nyelv annyira komplex, hogy ezt megkövetelei (ilyen nyelv az *ALGOL*). Többmenetes fordítást kell alkalmazni akkor is, ha a memória korlátozott mennyiségben áll rendelkezésre, mivel ez a módszer kevesebb memóriát követel, mint az egymenetes.

A többmenetes fordítás esetén lehetőség van optimalizálásra, viszont hosszabb ideig tart, és nagyobb a helyigénye.



c. Reguláris nyelvek

Egy Σ abc feletti reguláris kifejezések halmaza a (Σ unio {ø,(,),+,*})* halmaz legszűkebb olyan U részhalmaza, amelyre az alábbi feltételek teljesülnek:

- Az ø szimbólum eleme U-nak
- Minden a eleme Σ-ra az a eleme U-nak
- Ha R_1, R_2 eleme U, akkor $(R_1)+(R_2)$, $(R_1)(R_2)$ és $(R_1)^*$ is elemei U-nak

Az R reguláris kifejezés által meghatározott (reprezentált) |R| nyelvet a következőképpen definiáljuk:

- Ha R=ø, akkor R üres nyelv
- Ha R=a, akkor |R|={a}
- Ha $R=(R_1)+(R_2)$, akkor $|R|=|R_1| \cup |R_2|$
- Ha $R=(R_1)(R_2)$, akkor $|R|=|R_1||R_2|$
- Ha $R=(R_1)^*$, akkor $|R|=|R_1|^*$

Egy $L \leq \Sigma^*$ reguláris, ha van olyan Σ feletti R reguláris kifejezés, melyre |R|=L. A prioritási sorrend megállapodás: *, konkatenáció, +, valamint a konkatenáció és az unió asszociatív.

Bizonyítható, hogy minden véges nyelv egyben reguláris is. Egy tetszőleges Σ feletti L reguláris nyelv generálható reguláris nyelvtannal. Minden reguláris nyelv felismerhető automatával.