1.

a. Logikai műveletek, Neumann-elv

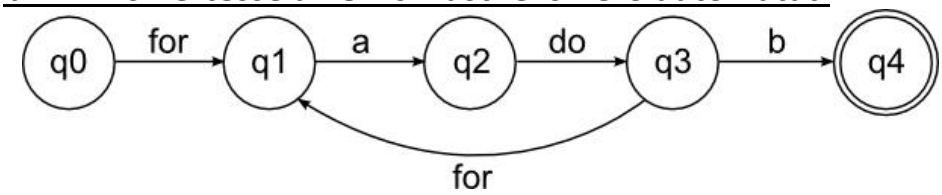
Logikai műveletek: ítéletkalkulus ítéletein definiált műveletek, amellyel az ítéletekből újabb összetett ítéleteket alkothatunk.
Egyváltozós logikai műveletek: igaz, hamis, A, ¬A. Az első kettő konstans, így az valójában nullaváltozós művelet.
Kétváltozós logikai műveletek:
- *Konjunkció* ($\wedge$): Logikai és. Eredménye csak akkor igaz, ha A és B is igaz.
  Ellentettje a NAND. ($\uparrow$)
- *Diszjunkció* ($\vee$): Logikai vagy (megengedő vagy). Eredménye akkor igaz, ha A vagy B vagy A és B igaz.
  Ellentettje a NOR. ($\downarrow$)
- *Implikáció* ($\rightarrow$): Logikai ha. Eredménye csak akkor hamis, ha A→B esetén A igaz, B hamis.
- *Ekvivalencia* ($\leftrightarrow$): Kölcsönös függőség, csak akkor ad igaz eredményt, ha mindkét változója igaz, vagy mindkettő hamis. Ellentettje a XOR(kizáró vagy).

Az operátorok precedenciája: ¬, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$

A Neumann - elvek
1. A számítógép legyen soros működésű: A gép az egyes utasításokat egymás után, egyenként hajtja végre.
2. A számítógép a kettes számrendszert használja, és legyen teljesen elektronikus: A kettes számrendszert és a rajta értelmezett aritmetikai, ill. logikai műveleteket könnyű megvalósítani kétállapotú áramkörökkel
(pl.: 1- magasabb feszültség, 0 - alacsonyabb feszültség)
3. A számítógépnek legyen belső memóriája: A számítógép gyors működése miatt nincs lehetőség arra, hogy minden egyes lépés után a kezelő beavatkozzon a számítás menetébe. A belső memóriában tárolhatók az adatok és az egyes számítások részeredményei, így a gép bizonyos műveletsorokat automatikusan el tud végezni.
4. A tárolt program elve: A programot alkotó utasítások kifejezhetők számokkal, azaz adatként kezelhetők - ezek a belső memóriában tárolhatók, mint bármelyik más adat. Ezáltal a számítógép önállóan képes működni, hiszen az adatokat és az utasításokat egyaránt a memóriából veszi elő.
5. A számítógép legyen univerzális: A számítógép különféle feladatainak elvégzéséhez nem kell speciális berendezéseket készíteni. Turing angol matematikus bebizonyította, hogy az olyan gép, amely el tud végezni néhány alapvető műveletet, akkor az elvileg bármilyen számítás elvégzésére is alkalmas.

A gépnek 5 alapvető funkcionális egységből kell állnia:
- a vezérlő egység (control unit),
- az aritmetikai és logikai egység (ALU),
- a tár (memory), ami címezhető és újraírható tároló-elemekkel rendelkezik, továbbá
- a ki/bemeneti egységek (Input/Output - I/O).

b. Ismertesse a FOR ciklust felismerő automatát.



c. Szintaktikus elemzés

Fordítási folyamat: Forrásprogram->Forrás-kezelő (source handler)->Lexikális elemző (scanner)->Szintaktikus elemző (parser)->Szemantikus elemző (semantic analizer)->Belső reprezentáció->Kódgenerátor (code generator)-> Optimalizáló (optimizer)->Kód-kezelő (code handler)->Tárgyprogram
Bemenete: szimbólumsorozat *Kimenete*: szintaxisfa, szintaktikus hibák *Feladata*: a program szerkezetének felismerése, a szerkezet ellenőrzése: megfelel-e a nyelv definíciójának?
A szintaktikus elemzőnek a feladata a program struktúrájának a felismerése. A szintaktikus elemző működésének az eredménye lehet például az elemzett program szintaxisfája vagy ezzel ekvivalens struktúra. Bemenete egy szimbólumsorozat, eredménye pedig a szintaktikusan elemzett program, és ha vannak, akkor a szintaktikai hibák.
A szintaxist nagyobb részében környezetfüggetlen grammatikával, kisebb részét környezetfüggő vagy attribútum grammatikával lehet leírni.
Szintaktikus elemzés = A környezetfüggetlen grammatikával leírható tulajdonságok vizsgálata.

2.

a. Bitmozgató műveleteket, Neumann-ciklus.

Léptetés (SHIFT)
A bitek értékeit egy helyi értékkel balra vagy jobbra léptetjük. Jobbra léptetés esetén a balszélső bit értéke vagy nulla, vagy az előjelbittel megegyező lesz, a jobb oldali bit értéke egy átviteli bit lesz, amit a gép eltárol. Balra léptetés esetén mindig nulla lesz a jobb oldali bit értéke, a balról kilépő érték pedig átvitelként jelenik meg.
A fixpontos számoknál a balra léptetés kettővel szorzásnak, a jobbra léptetés kettővel osztásnak felel meg.
Forgatás (ROTATE)
Balra rotálás esetén a kilépő bit vagy a legkisebb helyi értékre kerül, vagy eltárolódik átvitelként, és az előzőleg eltárolt átviteli bit kerül a legkisebb helyi értékre. Jobbra rotálás esetén a kilépő bit vagy a legmagasabb helyi értékre kerül, vagy eltárolódik átvitelként, és az előzőleg eltárolt átviteli bit kerül a legnagyobb helyi értékre.

ROL művelet kiinduló érték: 11001010 ROL utáni érték :10010101
carry értéke: 1, mivel a byte bal szélén kicsúszó bit értéke 1!A kiforgó bit beíródik a carry-be, és a byte másik szélén befordul !
ROR művelet kiinduló érték: 01001011 ROR utáni érték : 10100101
carry értéke: 1, mivel a byte bal szélén kicsúszó bit értéke 1! A kiforgó bit beíródik a carry-be, és a byte másik szélén befordul !
RCL művelet
Ha carry=0 kiinduló érték: 01001011 RCL utáni érték : 10010110
carry értéke: 0, mivel a byte bal szélén kicsúszó bit értéke 0!
Ha carry=1 kiinduló érték: 01001011 RCL utáni érték : 10010111
carry értéke: 0, mivel a byte bal szélén kicsúszó bit értéke 0!
RCR művelet
Ha carry=0
kiinduló érték: 01001011 RCR utáni érték : 00100101
carry értéke: 1, mivel a byte jobb szélén kicsúszó bit értéke 1!Ha carry=1
kiinduló érték: 01001011 RCR utáni érték : 10100101
carry értéke: 1, mivel a byte jobb szélén kicsúszó bit értéke 1!
SHL művelet
SHL [8 bites regiszter],1 - nél:

kiinduló érték: 01001010 SHL utáni érték :10010100
carry értéke: 0, mivel a byte bal szélén kicsúszó bit értéke 0!

Ez az utasítás egyel balra tolja a megadott regisztert. Ez utasítás megegyezik 2 hatványával való szorzással, ha például azt írjuk, hogy :

```
mov AX,3
shl AX,2
```

Akkor az "AX"-ben 12 lesz. [3*2^2]
SHR művelet
SHR [8 bites regiszter],1 - nél:

kiinduló érték: 01001010 SHR utáni érték :00100101
carry értéke: 0, mivel a byte jobb szélén kicsúszó bit értéke 0!

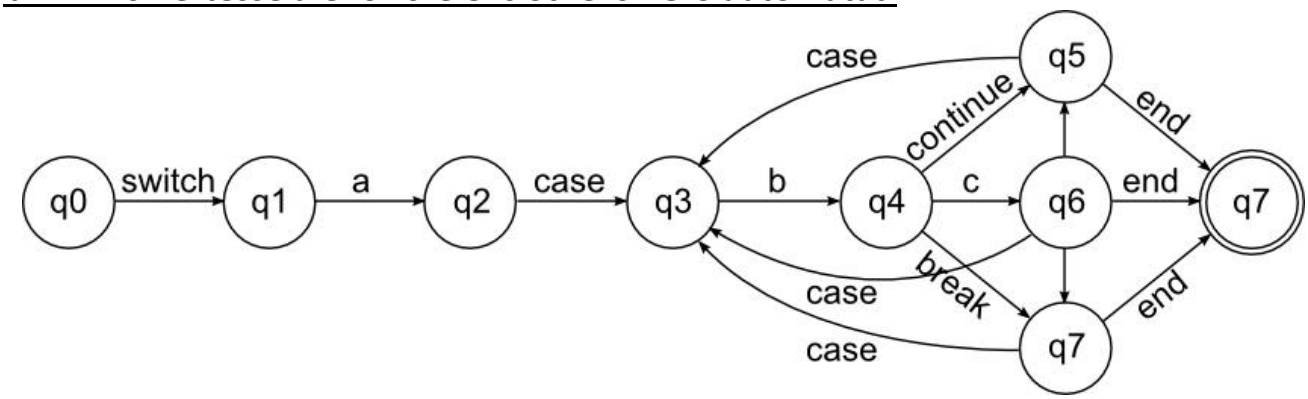
Ez az utasítás egyel jobbra tolja a megadott regisztert. Ez utasítás megegyezik 2 hatványával való osztással, ha például azt írjuk, hogy :

```
'MOV AX,4'
'SHR AX,2'
```

Akkor az "AX"-ben 1 lesz. [4/2^2]Vigyázzunk, mert ez az utasítás NEM figyel az előjelre!
SAL művelet
Ez elméletileg megegyezik az SHL-lel, csak figyel az előjelre is, azonban az utasításkódja ugyanaz, mint az shl-nek.
SAR művelet

Ez az utasítás eggyel jobbra tolja a megadott regisztert. Ez utasítás megegyezik 2 hatványával való osztással, és figyel az előjelre is! Magyarul ha a legfölső bit 1 volt, akkor a művelet során 1-es csúszik be felülről..

b. Ismertesse a CASE szelekciót felismerő automatát.



c. Az LL(k) grammatika

Az egyszerű LL(1) grammatika

Definició: A G grammatikát egyszerű LL(1) grammatikának nevezzük, ha

1. ε-mentes
2. A helyettesítési szabályok jobboldala terminális szimbólummal kezdődik.
3. Alternatívák esetén a jobboldalak kezdő terminális páronként különbözőek, azaz $A \rightarrow a1\alpha1 | a2\alpha2 | \ldots | ak\alpha k$ ahol $ai \neq aj$, ha $i \neq j$ $(1 \leq i, j \leq k)$.

LL(k) grammatika: a levezetés tetszőleges pontján a szöveg következő k terminálisa meghatározza az alkalmazandó levezetési szabályt

Az $S \Rightarrow^* wx$ legbaloldalibb levezetés építése során eljutunk a $S \Rightarrow^* wA\beta$ mondatformáig és az $A\beta \Rightarrow^* x$-et szeretnénk elérni, akkor az A-ra alkalmazható A > a helyettesítést egyértelműen meghatározhatjuk az x első k db szimbólumának előre olvasásával (ekkor és csak ekkor LL(k) nyelvtanról beszélhetünk).

$FIRST_k(\alpha)$: az α mondatformából levezethető terminális sorozatok k hosszúságú kezdőszeletei

$FIRST_k(\alpha) = \{x \mid \alpha \Rightarrow^* x\beta \text{ és } |x| = k\} \cup \{x \mid \alpha \Rightarrow^* x \text{ és } |x| < k\}$

A G grammatika LL(k) grammatika, ha tetszőleges $S \Rightarrow^* wA\beta \Rightarrow w\alpha1\beta \Rightarrow^* wx$ és $S \Rightarrow^* wA\beta \Rightarrow w\alpha2\beta \Rightarrow^* wx$ levezetés párra $FIRSTk(x)=FIRSTk(y)$ esetén $\alpha1=\alpha2$. Eszerint tehát ha egy grammatika LL(k) grammatika, akkor a már elemzett w utáni k darab terminális szimbólum az A-ra alkalmazható helyettesítési szabályt egyértelműen meghatározza.

3.

a. Feltétel nélküli vezérlésátadás, a processzor utasításrendszere, operandusok, címzési módok.

Feltétel nélküli vezérlésátadás: Feltétel nélküli ugrásnál az utasításban szereplo címmel tölti fel a processzor az utasításszámláló regiszter tartalmát, amely a következo utasítás címe lesz és a program innen folytatódik.

(Assembly-ben: JMP címke)

- nem elegáns használni
  - több nyelvben már nem is létezik ez az utasítás, vagy csupán végtelen ciklusból való kiugrásra lehetséges a használata
- a probléma vele többrétű
  - átláthatatlanná teheti a programstruktúrát, hibakeresést megnehezíti
  - a fordítóprogram nem tudja miatta jól meghatározni a program erőforrás- vagy memóriaigényét

A processzor utasításrendszere:

Minden számítógépnek van egy belső memóriája az éppen futó programok és az adatok tárolására. A PC processzorok memóriája byte szervezésű, azaz minden byte-nyi memóriának van egy memória címe. Bármely két szomszédos byte egy 16 bites szót alkot. A gépi kódú utasítások négy részből állnak, méretük függ az utasítástól és a címzési módtól.

Az általános utasításszerkezet: Prefixum|Operáció kód|Címzési mód|Operandus

1. A prefixum módosítja az utasítás értelmezését, pl. ezzel írható elő az ismétlések száma, vagy a megszakításkérés tiltása. Használata nem kötelező.
2. Az operáció kód adja meg, hogy a processzornak milyen műveletet kell végrehajtania. Használata kötelező.
3. A címzési mód az operandusok értelmezését adja meg. Nem minden utasításban található meg.
4. Az operandus lehet konstans, cím vagy címzéshez használt érték. Hossza változó, használata nem kötelező.

Az assembly utasítások pontosan megfelelnek egy gépi kódú utasításnak, azaz minden assembly utasításból egy gépi kód lesz.
Operandusok: Az Intel processzorokban egy utasítással csak egy memóriahely címezhető meg. Azaz, ha egy utasítás két operandusú, akkor csak az egyik lehet memóriacím, a másik regiszter kell, hogy legyen.
Regiszteroperandus: Az utasítás paraméterei regiszterek, amelyekben a művelethez szükséges adatok vannak. Az utasítás hossza a regisztermérettől függ. Példa: MOV AX,BX
Közvetlen operandus: Az operandus az utasítás kódját tartalmazza. Példa: MOV AX,2
Címzési módok:
Direkt memóriacímzés: Itt az operandus egy, a Data szegmensben előre deklarált változó.

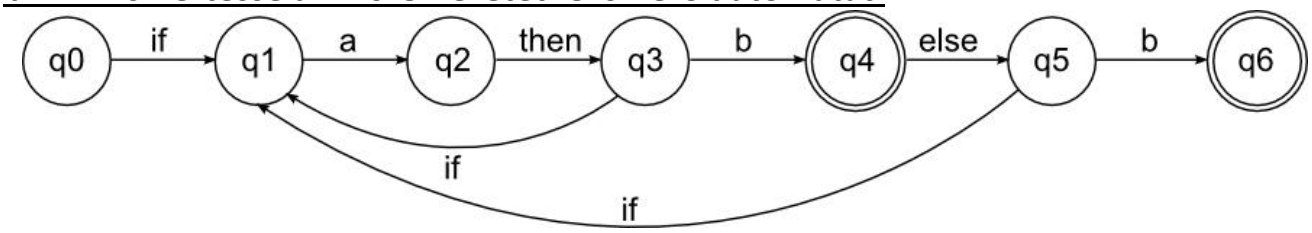
```
.DATA
 adat DB "A"
.CODE
 MOV AX, adat
```

Indirekt memóriacímzés: Itt az operandus az adat címét tartalmazza, nem az értékét.
Regiszter indirekt címzés: Példa: MOV AX,[BX] ;AX-be tölti a BX által megcímzett memória tartalmát
Indexelt, bázis-relatív címzés

```
.DATA
 adat DB 10h,20h
.CODE
 MOV AX,adat[BX] ;AX-be tölti az adat+BX címen lévő adatot
```

Bázis plusz index címzés
Példa: MOV AX,[BX][DI] ;AX-be tölti a BX+DI címen lévő adatot
Bázis plusz relatív címzés
Példa: MOV AX,adat[BX][DI] ;AX-be tölti az adat+BX+DI címen lévő adatot
Ez a megoldás használható többdimenziós tömbök kezelésére.

b. Ismertesse az IF szerkezetet felismerő automatát.



c. Determinisztikus véges automaták.

Automata: egy olyan konstrukció, mely egy input szóról képes eldönteni, hogy az helyes-e vagy sem.
1. Létezik egy input szalagja, amely cellákra van osztva, és minden cellában egy-egy karakter van. Az input szalag általában véges és olyan hosszú, mint az input szó.
2. Az input szalaghoz egy olvasófej tartozik, amely mindig egy cella fölött áll és ezen cellából képes kiolvasni a karaktert. Az olvasófej lépked a szalag cellái között egyesével, balra-jobbra.
3. Létezik egy output szalagja, ami cellákra van osztva, és minden cellában egy-egy karakter lehet. Az output szalag lehet véges vagy végtelen. Azokban a cellákban,amelyekbe még nem írt semmit az automata, egy speciális karakter, a BLANK jel áll.
4. Az output szalaghoz egy író-olvasó fej tartozik. Ennek segítségével az automata egy jelet írhat vagy olvashat az aktuális cellából. Az író-olvasó fej léptethető a cellák fölött balra-jobbra.
5. Az automatának belső állapotai vannak, melyek között az automata lépkedhet egy megadott séma alapján. Az automata mindig pontosan egy állapotban van (aktuális állapot).

Véges automaták:. Nincs output szalagjuk, sem író-olvasó fej. Az input szalag hossza véges, épp olyan hosszú, mint az input szó (ugyanannyi cellából áll, amennyi a szó hossza).

Egy M(Q, Σ, Δ, q0, F) ötöst véges automatának nevezzük, ahol
1. Q az automata állapotainak véges halmaza
2. Σ az elemzendő jelsorozat
3. Δ az automata mozgási szabályainak véges halmaza
4. q0 az induló állapot, q0∈Q
5. F az elfogadó állapotok halmaza, F ⊂ Q

A véges automaták egy irányított gráffal szemléltethetőek, és az egyes csomópontjai felelnek meg az automata állapotainak. Amennyiben egy karakter beolvasásának hatására az automata egyik állapotból egy másik állapotba megy át, akkor a két állapotnak megfelelő csomópontokat egy, az eredeti állapotból az új állapotba mutató, és az olvasott karaktert mint nevet (színt) viselő éllel kötjük össze.
Mozgási szabályok: megmondják, hogy egy adott állapotban (A), egy adott karakter (a) beolvasásának hatására milyen új állapotot vesz fel az automata: Δ(A, a) = B. A mozgási szabályok összessége tulajdonképpen egy leképezés, amely az automataállapotok és az alfabéta karaktereinek direkt szorzatából álló halmazt képezi le az automataállapotok halmazára: Q x Σ ⇒ Q.
Determ. aut.: Ha minden állapot–karakter párhoz legfeljebb egy mozgási szabály tartozik, akkor az automata működése egyértelműen meghatározott, az automata determinisztikus.

4.

a. Összeadás, kivonás és szorzás műveletek, memóriaszegmensek kezelése.

Aritmetikai műveletek: A számítógépek minden műveletet bináris formában végeznek el. A gépek fix hosszúságú számokkal dolgoznak.
Az A és B szimbólumokkal változókat jelölünk, az S az eredmény, a C pedig a maradék vagy átvitel jelölése.
Összeadás: A bináris összeadás hasonló, mint a decimális számoknál: összeadjuk az adott helyi értéken a kész számot (A+B=S), és ha van maradék (C), akkor azt hozzáadjuk a magasabb helyi érték 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. 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: 00010001 => Kettes komplemens vétele: hozzáadunk 1-et: 00010010 (és ezt adjuk hozzá az eredeti számhoz, így a kivonott értéket fogjuk kapni)
Szorzás: A bináris szorzás úgy végződik el, mint a decimális: 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	=	1

Az eredmény az A és a B értékkel elvégzett AND logikai művelet.
Byte-os szorzásnál legalább két byte-nyi adatterületre van szükség az eredmény tárolásához, így az eredmény egy kétbyte-os regiszterbe kerül. Ha kétbyte-os adatokkal dolgozunk, akkor négybyte-os lesz a szorzat.

b. A többmenetes fordítók működése.

Többmenetes fordító: a fordítási fázisok több különböző menetben futnak le. Ilyenkor szükség van közbenső programformák tárolására, ezek az egyes menetek outputjai és a másikak inputjai. Sok esetben a többmenetes fordítás az egyetlen lehetőség a nyelv komplexitása miatt.
Pl.: **Forrásnyelvű program** => Lexikális elemzés => Szintaktikai és szemantikai elemzés => **Közbenső programforma**
majd második menet: **Közbenső pf.** =>Kódgenerálás és kódszelektálás => **Tárgynyelvű program**

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 R1, R2 eleme U, akkor (R1)+(R2), (R1)(R2) és (R1)* 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=(R1)+(R2), akkor |R|=|R1|∪|R2|
- Ha R=(R1)(R2), akkor |R|=|R1||R2|
- Ha R=(R1)*, akkor |R|=|R1|*

Egy L ≤ Σ* 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.

5.

a. Konstansok definiálása, STACK szerepe a vezérlésátadásban.

Konstansok: A konstansok általában négyféle számrendszerben is megadhatók: binárisan, oktálisan, decimálisan és hexadecimálisan. Assembly esetén a szám után írt betűvel lehet jelezni az egyes számrendszereket:

- bináris: B vagy Y betű
- oktális: O vagy Q betű
- decimális: D vagy T betű
- hexadecimális: H – hogy a változókkal ne keverjük össze, a szám elé „vezető 0” kerül, pl.: 0FFh

Alapértelmezett számrendszer a decimális. Átállítása a .RADIX n (n a számrendszer alapja) direktíva alkalmazásával történik, ekkor a számok végéről az adott betű elhagyható (pl.: .RADIX 16 esetén a H).
A karakter-konstansok megadásánál használhatjuk a karakter ASCII kódja, de magát a karaktert is. (pl.: MOV AL, „A” MOV AL, 65)
STACK: A stack (verem) egy olyan – az operációs rendszer által kijelölt memóriaterület – ahova közvetlen utasításokkal lehet adatot elmenteni, és onnan visszaolvasni. Itt átmeneti adatok tárolhatóak, ilyen lehet szubrutin hívásakor a visszatérési cím, vagy egyes függvények, eljárások paraméterei. Az operációs rendszer a megszakításokkor ide menti a futásához szükséges regiszterek tartalmát.
A stack egy LIFO (last in, first out) szervezésű memória. Amikor valami bekerül a verembe, a veremmutató (a verem tetejére, az utoljára betett adat memóriacímére mutat) értéke kettővel csökken, ha valami kikerül onnan, kettővel nő. Ez a fajta memóriaszervezés lehetővé teszi szubrutinok hívását, tetszőleges mélységig.
A CALL szubrutin hívás elmenti a stack tetejére a következő (azaz a CALL utáni) utasítás címét, és ezt a címet tölti be az IP-be a RET utasítás. A verem tetején tárolt adat teljes címe az SS:SP regiszterpárban található.
Assemblyben a PUSH utasítással lehet adatot rakni a verembe, POP utasítással kivenni.

b. Direktívák csoportosítása, működésük.

Olyan parancsok, amelyek vagy a fordítást, vagy pedig a futtatást befolyásolják. A direktívákat és az operandusaikban lévő szimbólumokat is a lexikális elemző határozza meg, ez általában előfeldolgozó modul segítségével történik.
Assembly példa: .STACK, .DATA, .CODE (a program megfelelő szegmenseire utalnak); END (programvég)
Pascal példák: kapcsoló direktívák (kétféle lehetőségük van, + jelre bekapcsolnak, - jelre ki). Pl.: {$B-} csak addig vizsgálja a kifejezést (and és or), amíg nem dönthető el egyértelműen; {$D+} debug infókat fűz a fordító a programhoz; {$I+} I/O műveletek automatikus ellenőrzése - kikapcsolva IOResult-tal ellenőrizhető és kezelhető
Másik Pascal példa: feltételes fordítás: csak akkor fordítja le, ha megadott feltétel teljesül. {$IF feltétel}...{$ENDIF}

c. LR(1) grammatikák és elemzések

Alulról felfelé elemzéseknél *(8/C TÉTEL)* a terminális szimbólumokból kiindulva haladnak a kezdőszimbólum felé. Ezek az elemzések egy vermet használnak, és a vizsgált módszerek léptetés-redukálás típusúak, azaz az elemző mindig egy redukciót próbál végezni, ha ez nem sikerül, akkor az input szimbólumsorozat következő elemét lépteti. Ezek az elemzések a legjobboldalibb levezetés "inverzét" adják. Az elemzés alapproblémája a mondatformák nyelének a meghatározása.
Első lépésben a grammatika szabályaiból egy táblagenerátorral elkészítünk egy elemző táblázatot, majd az elemző program a táblázattal és egy veremmel tudja elemezni a szöveget. Egy G' kiegészített grammatikát LR(k) grammatikának nevezünk, ha
S'→αAw→αβw
S'→*γBx→γδx = αβy
FIRSTk(w) = FIRSTk (y) esetén α=γ, A=B, x=y. Ekkor αβw mondatformában, k szimbólumot előreolvasva a w első szimbólumától egyértelműen meghatározható, hogy β a nyél, és az A→β szabály szerint kell redukálni. Minden LR(k) grammatikához létezik egy vele ekvivalens LR(1) grammatika. Ha egy αβx mondatforma nyele β, akkor az αβ prefixeit az αβx járható prefixeinek nevezzük.

Működés: A szimbólumokat olvasási sorrendben egy verembe helyezzük el, közben vizsgáljuk, hogy nem alakult-e ki egy olyan csoport, amely a levezetési szabály jobboldala, vagyis egy nyél. Amennyiben a csoport nyél, akkor helyébe a szabály baloldalán álló nemterminálist írjuk.
Az elemzés menete: A verem szimbólumpárokat tartalmaz, az első elemben egy terminális vagy nemterminális szimbólumot tárolunk, a másodikban az automata állapotának a sorszámát. A verem kezdeti tartalma legyen (#,0). Ha az automata egy léptetést hajtott végre, akkor a beolvasott szimbólum és az új állapot sorszáma kerül a verembe. A redukció állapotban, ha az A→α szabály szerint kell redukálni, akkor töröljük a verem |α| db. sorát, azaz 2*|α| elemét, írjuk a verem tetejére, az első elembe az A szimbólumot, lépjünk vissza az automatában a járható prefixszel bejárt úton |α| lépést. Határozzuk meg, hogy ebből az állapotból az A hatására melyik állapotba kerül az automata, és ezt az állapotsorszámot írjuk a verem tetején levő második elembe. Ha egy állapotátmenettem az S'→S szabály redukciójához jutunk, akkor az elemző az input szöveget elfogadta és megáll. Ha egy állapotból olyan input szimbólum hatására kell továbblépni, amelyhez a táblázatban nincs megadva állapotsorszám, akkor az elemzés szintaktikus hibát detektál, és megáll.

6.

a. Feltételes vezérlésátadás, a CPU fontosabb regiszterei és alkalmazásuk.

Az assembly nyelv egyik szegényes tulajdonsága a vezérlési szerkezetek hiánya. Lényegében csak az alábbi vezérlési szerkezetek találhatóak meg:

- Szekvencia: a program utasításainak végrehajtása a memóriabeli sorrend alapján történik.
- Feltétel nélküli vezérlésátadás (ugró utasítás): a program folytatása egy másik memóriabeli pontra tevődik át, majd attól a ponttól kezdve a végrehajtás újra szekvenciális.
- Feltételes vezérlésátadás (feltételes ugró utasítás): mint az egyszerű ugró utasítás, de az ugrást csak akkor kell végrehajtani, ha az előírt feltétel teljesül.
- Visszatérés az ugró utasítást követő utasításra (azon helyre, ahonnan az ugrás történt).

Regiszterek: A regiszterek méretével jellemezhető egy CPU, azaz lehet 8, 16, 32 stb bites. A nagyobb méret (elméletileg) gyorsabb processzort jelent, de csak egyre nagyobb adatmennyiségeknél igaz ez.

1. Általános regiszterek

- Akkumulátorregiszter (Jelölése: AX, alsó bitje: AL, felső bitje: AH) :

Szerepe van a szorzás, osztás és I/O utasításoknál.

- Bázisregiszter (BX,BL,BH)

Szorzás és osztástól eltekintve minden művelethez használható, általában az adatszegmensben tárolt adatok báziscímét tartalmazza.

- Számlálóregiszter (CX,CL,CH)

Szorzás és osztás kivételével minden művelethez használható, általában ciklus, léptető, forgató és sztring utasítások ciklusszámlálója.

- Adatregiszter (DX,DL,DH)

Minden művelethez használható, de fontos szerepe van a szorzás, osztás és I/O műveletekben.

2. Vezérlő regiszterek

- Forrás cím (Jelölése: SI)

A forrásadat indexelt címzésére. Szorzás és osztás kivételével minden műveletnél használható.

- Cél cím (Jelölése: DI)

A céladat indexelt címzésére. Kitüntetett szerepe van a sztring műveletek végrehajtásában. Az SI regiszterrel együtt valósítható meg az indirekt és indexelt címzés. Szorzás és osztás kivételével minden műveletnél használható.

- Stack mutató (Jelölése: SP)

A verembe utolsóként beírt elem címe. A mutató értéke a stack műveleteknek megfelelően automatikusan változik.

- Bázis mutató (Jelölése: BP)

A verem indexelt címzéséhez. Használható a stack-szegmens indirekt és indexelt címzésére. Más műveletben nem javasolt a használata.

- Utasításmutató (Jelölése: IP)

A végrehajtandó utasítás címét tartalmazza, mindig a következő utasításra mutat. Az utasítás beolvasása közben az IP az utasítás hosszával automatikusan növekszik.

3. Szegmensregiszterek

Ezek a regiszterek tárolják a különböző funkciókhoz használt memória- és szegmenscímeket.

- Kódszegmens (Jelölése: CS)

Az utasítások címzéséhez szükséges, az éppen futó programmodul báziscímét tartalmazza. Minden utasításbetöltés használja. Tartalma csak vezérlésátadással módosulhat.

- Veremszegmens (Jelölése: SS)

A verem címzéséhez használatos, a stack-ként használt memóriaterület báziscímét tartalmazza.

- Adatszegmens (Jelölése: DS)

Az adatterület címzéséhez kell, az adatszegmens bázis címét tartalmazza.

b. Magas szintű programnyelvek fordítása, compiler, interpreter.

Ha a forrásnyelv egy magas szintű nyelv, akkor a forráskód és a gépi kód között jelentős a különbség.
forrásnyelvű program => compiler => tárgyprogram => végrehajtás (adatok=>eredmény)
Fordítóprogram (compiler): olyan számítógépes program, amely valamely programozási nyelven írt programot képes egy másik programozási nyelvre lefordítani. A fordítóprogramok általánosan forrásnyelvi szövegből állítanak elő tárgykódot. A fordítóprogramok feladata, hogy nyelvek közti konverziót hajtsanak végre. A fordítóprogram a forrásprogram beolvasása után elvégzi a lexikális, szintaktikus és szemantikus elemzést, előállítja a szintaxis fát, generálja, majd optimalizálja a tárgykódot.
Compiler feladatai:

• Analízis: a forrásnyelvű program karaktersorozatát részekre bontja, még a szintézis az egyes részeknek megfelelő tárgykódokból építi fel a program teljes tárgykódját.

- Lexikális elemző (karaktersorozat) (szimbólumsorozat, lexikális hibák): a karaktersorozatban meghatározza az egyes szimbolikus egységeket, a konstansokat, változókat, kulcs szavakat és operátorokat. A karaktersorozatból szimbólumsorozatot készít, ki kell szűrnie a szóköz karaktereket a kommenteket, mivel ezek tárgykódot nem adnak. A magas szintű programnyelvek utasításai általában több sorban írhatók a lexikális elemző feladata, egy több sorba írt utasítás összeállítása is.
  - Szimbólumtábla létrehozása (szimbólum típusa, szimbólum címe)
  - Szóköz karakterek és kommentek kiszűrése
  - Többsoros utasítások összeállítása
- Szintaktikus elemző (szimbólumsorozat) (szintaktikusan elemzett program, hibák): a program struktúrájának a felismerése. A szintaktikus elemző működésének az eredménye lehet például az elemzett program szintaxisfája vagy ezzel ekvivalens struktúra.
  - Szimbólumok helyének ellenőrzése
  - Ellenőrzi, hogy a szimbólumok sorrendje megfelel-e a programnyelv szabályainak
  - Szintaxisfa előállítása
- Szemantikus elemző (szintaktikusan elemzett program) (analizált program, hibák): feladata bizonyos szemantikai jellegű tulajdonságok vizsgálata. A szemantikus elemző feladat például az a+b kifejezés elemzésekor az, hogy az összeadás műveletének a felismerésekor megvizsgálja, hogy az „a” és a „b” változók deklarálva vannak-e, azonos típusúak-e és hogy van-e értékük.
  - Konstansok, változók érték- és típusellenőrzése
  - Aritmetikai kifejezések ellenőrzése

• Szintézis:

- Kódgenerátor (analizált program) (tárgykód): a tárgykód gépfüggő, generációs rendszertől függő, a leggyakrabban assembly vagy gépi kódú program.
- Kódoptimalizáló (tárgykód) (tárgykód): A kódoptimalizálás a legegyszerűbb esetben a tárgykódban lévő azonos programrészek felfedezését és egy alprogramba való helyezését vagy a hurkok ciklus változásától független részeinek megkeresését és a hurkon kívül való elhelyezését jelenti. Egy jó kódoptimalizálónak jobb és hatékonyabb programot kell előállítania, mint amit egy gyakorlott programozó tud elkészíteni.

Interpreter: Hardveres, vagy virtuális gép, mely értelmezni képes a magas szintű nyelvet, vagyis melynek gépi kódja a magasszintű nyelv. Ez egy kétszintű gép, melynek az alsó szintje a hardver, a felső az értelmező és futtató rendszer programja. *Feladatai*: Beolvassa a következő utasítást, és eldönti, hogy az utasításkészlet melyik utasítása.

• Ellenőrzi, hogy az adott utasítás szintaktikailag helyes-e, az átadott paraméterek típusa, esetleg mérete megfelel-e az utasítás kívánalmainak

• Ha hibátlan, akkor meghívja az utasításhoz tartozó előre elkészített kódrészletet

• Figyeli, hogy a végrehajtás során nem jön-e létre valamilyen hiba. Ha hibára fut a rendszer, akkor hibakódot kell generálnia. Ha hibátlan a végrehajtás, akkor veszi a következő utasítást

c. Chomsky-féle nyelvosztályok

Adott egy G=(T,N,S,P) grammatika, és
$\alpha, \beta, \gamma \in (T \cup N)^*$ mondatformák, lehetnek ε értékűek is
$\omega \in (T \cup N)^+$ mondatforma, de nem lehet ε
$A,B \in T$ terminális jelek

a, b ∈ N nem terminális jelek

Minden típusra igaz, hogy akkor hívunk egy nyelvet az adott típusúnak, ha van olyan grammatika, ami az adott nyelvet generálja.
Reguláris (3-típusú) nyelvek: itt egy szabály kétféle alakú lehet: A → a, vagy A → aB.
Környezetfüggetlen (2-típusú) nyelvek: egy helyettesítési szabály nem függ a környezetétől, sőt, környezete sem lehet, azaz A → ω, és megengedett az S → ε.
Környezetfüggő (1-típusú) nyelvek: egy helyettesítési szabály csak bizonyos környezetben alkalmazható, azaz βAγ → βωγ, és megengedett az S → ε.
Általános (0-típusú) nyelvek: helyettesítési szabályokra nincs megszorítás, azaz βAγ → α.

Az egyes típusok közötti összefüggés: {reguláris nyelvek} ⊆ {környezetfüggetlen nyelvek} ⊆ {környezetfüggő nyelvek} ⊆ {általános nyelvek}. Hasonló összefüggés igaz a grammatikákra is: {reguláris grammatikák} ⊂ {környezetfüggetlen grammatikák} ⊂ {környezetfüggő grammatikák} ⊂ {általános grammatikák}.

7.

a. Ciklusok (iterációk), az Assembly és a C nyelv kapcsolata.

Akkor szoktunk ciklusokat szervezni, ha van egy tevékenységünk és azt sokszor kell elvégezni. Beszélünk determinisztikus és nem determinisztikus ciklusokról. A *determinisztikus ciklus* esetében tudjuk, hogy egy adott tevékenységet hányszor kell elvégezni. Magas szintű nyelvekben ezt "for" ciklusnak hívják.
LOOP utasítás: ezt a ciklusmag végére kell tenni és annyiszor ugrunk el vele a ciklus elejére, amennyit az ECX regiszterben beállítottunk.
nem determinisztikus ciklusok: Ezeknek két fajtája van: az elöltesztelő (while) és a hátultesztelő (do while). Az elöltesztelő ciklusnál a ciklusfejben van egy teszt, ha ez egy feltételt igaznak talál, akkor végrehajtódik a ciklusmag és visszatérünk a ciklusfejhez. Ezért hívjuk ezt while (amíg) ciklusnak. A hátultesztelő ciklusnál először lefut a ciklusmag, majd a cikluslábban egy teszt eldönti, hogy szükséges-e a ciklusmagot újra végrehajtani.
Egy MASM fordítónak megadható, hogy ha egy assembly rutint C programhoz írunk:
.MODEL SMALL,C
Emellett szükséges még a PUBLIC direktíva használata is, mely közli az assemblerrel, hogy az utána írt eljárást nyilvánossá kell tenni más programok számára. A PUBLIC miatt az assembler olyan információkat generál a LINK (szerkesztő) számára, amely lehetővé teszi a külső hozzáférést.

```
.MODEL SMALL,C
.CODE
PUBLIC clear_screen
clear_screen PROC
…
clear_screen ENDP
```

A C programok vermet használnak a paraméterek átadására, az assembly rutinokban viszont rendre regiszterekben történt az adatátadás.

b. Adatábrázolás, adattípusok.

Kétféle adatot különböztethetünk meg:

- A gépi számábrázolású adatok, ezeken végez műveleteket a számítógép
- Egyéb adatok, ezek állhatnak tetszőleges karakterekből. Ezeket a gép kódoltan ábrázolja, tárolja, és csak speciális esetben végez velük műveletet

Fixpontos számábrázolás
A legegyszerűbb számábrázolási mód, az egész számok ábrázolása és tárolása. Egy tetszőleges számot kettes számrendszerben, az annak megfelelő bithosszúságon kell tárolni. Törtek esetén rögzítjük a tizedespont helyét, ettől lesz fixpontos a számábrázolás. Ennek a speciális esete az egész számok ábrázolása, amikor is a tizedespontot az utolsó bit után rögzítjük. Megkülönböztetünk előjeles és előjel nélküli számábrázolást is: az előjelesnél az első bit az előjelbit, pozitív számoknál 0, negatívoknál 1 az értéke.
Fixpontos számoknál figyelni kell a túlcsordulásra, azaz hogy egy művelet eredménye ne legyen hosszabb, mint ami a rendelkezésre álló helyen elférhet, azaz 1 byte esetén ne legyen nagyobb 255-nél. A túlcsordult számjegyek

elvesznek, a maradékkal való számolás pedig hibás eredményt adhat. A fixpont hátránya még, hogy nem lehet tetszőleges pontossággal ábrázolni az értékeket.

A lebegőpontos ábrázolás alapja, hogy a számok felírhatóak hatványkitevős alakban is: ±m×pk, ahol a p a számrendszer alapszáma (általában 2), az m a mantissza, a k pedig a karakterisztika. Ahhoz, hogy a felírás mindig egyértelmű legyen, az m mindig kisebb mint 1, és a tizedesponttól balra eső számjegy nem lehet 0. $1/p \leq m < 1$

c. Felülről-lefelé elemzés.

A kezdőszimbólumból (S), a szintaxisfa gyökeréből elindulva építjük fel a szintaxisfát. Célunk, hogy a szintaxisfa levelein az elemzendő szöveg terminálisai legyenek. Felülről-lefelé elemzéskor mindig a legbaloldalibb helyettesítéseket alkalmazzuk (ha $A \rightarrow \alpha \in P$, akkor az $x A \beta$ mondatforma legbaloldalibb helyettesítése $x\alpha\beta$, azaz $xA\beta \Rightarrow_{legbal} x\alpha\beta$). Végrehajtás az Earley algoritmussal:
Kiindulunk az S szimbólumból
Az S helyettesítésére az első olyan szabályt alkalmazzuk, amely bal oldalán S áll
Az így létrejött mondatforma legbaloldalibb nemterminálisára alkalmazzuk a szabályt
A létrejött mondatformára alkalmazzuk a 3. pontot mindaddig, míg
a mondatforma baloldalán lévő terminálisok megegyeznek az elemzendő szöveggel
a mondatformában van nemterminális szimbólum
Ha nincs több nemterminális és a mondatforma azonos az elemzendő szöveggel: VÉGE
Ha a mondatformában nincs több nemterminális, és nem egyezik meg a szöveggel, vagy a baloldalra kerülő terminálisok nem egyeznek meg a szöveg prefixével: VISSZALÉPÉS
Mindig az utoljára alkalmazott helyettesítést lépjük vissza:
Ha van következő helyettesítési szabály akkor azt hajtjuk végre, folytatás a 4.-től,
Ha S-hez jutunk vissza, és S-nek nincs több helyettesítési szabálya: VÉGE (nem mondat),
Ha a nemterminális nem S, és nincs több helyettesítési szabálya: egyet visszalépünk.

8.

a. Szoftveres megszakításkezelés terminál input-output, az Assembly és a Pascal nyelv kapcsolata.

Szoftver-megszakításnak egy program által kiválasztott megszakítást nevezünk. Ez vagy megkönnyíti a különféle hardvereszközök elérését, vagy az operációs rendszer egyes funkcióval is kapcsolatot teremthet.
Megszakításra INT utasítást használunk (egyoperandusú). Egy adott számú megszakítás sokszor sok szolgáltatás kapuját jelenti, ilyenkor a paramétereket regiszterekben kell tárolni. A visszatérő értékeket is általában regiszterekben kapjuk meg.

DOS-t az int 21h –val tud szöveget kiírni. (A 4Ch sorszámú funkciót már eddig is használtuk a programból való kilépésre. 09h dollárjellel lezárt stringet ír ki a megfelelő karakterpozícióba.) Billentyű kezelését az int 16h-n keresztül tudjuk elérni, a szolgáltatás számát itt is AH-ba kell rakni.:

```
MODEL SMALL
.STACK
ADAT SEGMENT
        Kerdes DB "Tetszik az Assembly? $"
        Igen DB "Igen.",0Dh,0Ah,'$'
        Nem DB "Nem.",0Dh,0Ah,'$'
ADAT ENDS
KOD SEGMENT
ASSUME CS:KOD,DS:ADAT
@Start:
        MOV AX,ADAT
        MOV DS,AX
        MOV AH,09h
        LEA DX,[Kerdes]
        INT 21h
```

```
@Vege:
        MOV AH,09h
        INT 21h
        MOV AX,4C00h
        INT 21h
        KOD ENDS
END @Start
```

*Pascal*ban használhatóak assembly kódok, közvetlenül a pascal kódba ágyazva. Az assembly kódot ASM jelöléssel kell nyitni, és END-el zárni. Assembly programmodulok futtatásakor nem kell verembe menteni az AX,BX,CX és DX regisztereket, de az SI, DI, BP, SP, CS, DS és SS-t igen.

Példa:

```
Function Osszeg(A,B: Byte):Byte; Assembler;
ASM
  MOV AL,A
  ADD AL,B
END;
```

b. Számrendszerek közötti konverzió.

A konverzió átalakítást jelent, ami esetünkben annyit tesz, hogy közreadunk egy olyan módszert, amely segítségével egyetlen lépésben megoldható bármilyen szám felírása az ismert számrendszerekben, mert valójában minden ábrázolt szám ugyanabból a bitkombinációból áll. A számítógép nyolcas számrendszer esetében triádokat (három bitből álló csoport), míg a tizenhatos számrendszerben tetrádokat (négy bitből álló csoport) képez (a képzés minden esetben jobbról-balra halad). (…)

c. Alulról-felfelé elemzés.

Alulról felfelé elemzéseknél a terminális szimbólumokból kiindulva haladnak a kezdőszimbólum felé. Ezek az elemzések egy vermet használnak, és a vizsgált módszerek léptetés-redukálás típusúak, azaz az elemző mindig egy redukciót próbál végezni, ha ez nem sikerül, akkor az input szimbólumsorozat következő elemét lépteti. Ezek az elemzések a legjobboldalibb levezetés ,,inverzét" adják. Az elemzés alapproblémája a mondatformák nyelének a meghatározása.

Szintaxisfa felépítésénél az elemzendő szimbólumsorozatból indulunk ki, megkeressük a mondatforma nyelét, amit helyettesítünk a hozzá tartozó nemterminális szimbólummal. A cél, hogy elérjük a grammatika kezdőszimbólumát, ami a szintaxisfa gyökérszimbóluma. A fa levelei pedig az elemzendő program terminális szimbólumai kell, hogy legyenek. Egy mondatforma legbaloldalibb egyszerű részmondatát a mondatforma nyelének nevezzük (egy mondatforma szintaxisfájában a legbaloldalibb, csak gyökérelemeket és leveleket tartalmazó részfa a mondatforma nyele).

Ha $A \rightarrow \alpha \in P$, akkor az $x A \beta$ mondatforma legbaloldalibb helyettesítése $x\alpha\beta$, azaz $xA\beta \rightarrow_{legbal} x\alpha\beta$. Ha $A \rightarrow \alpha$, akkor a βAx mondatforma legjobboldalibb helyettesítése $\beta\alpha x$, azaz $\beta Ax \rightarrow$legjobb $\beta\alpha x$.

9.

a. Adatátadás alprogramok között, makrók létrehozása, fordítása.

Adatátadás alprogramok között(paraméter átadás): Például van egy C főprogramunk és egy Assembly szubrutinunk, valamilyen módon adatot kell hogy cseréljenek, adatot kell tudniuk átadni egymásnak. Az assembly példaprogramjainkban az adatok átadására regisztereket használtunk. A C programok azonban a vermet (stacket) használják a paraméterek átadására. Azzal, hogy megadjuk az assemblernek, hogy C-hez írunk alprogramot, néhány beépített automatizmus egyszerűsíti a program írását.

A paraméterátadás egyszerűbb megértéséhez nézzünk egy egyszerű példát:

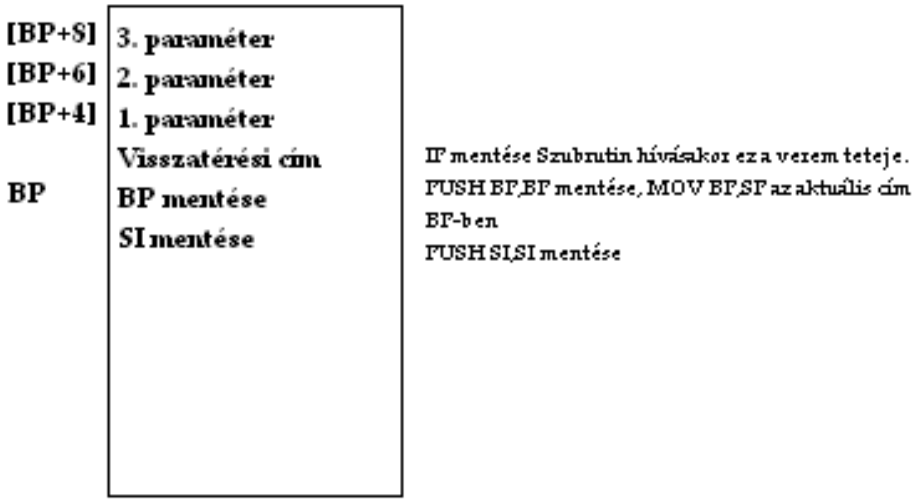
Legyen a C programunk a következő:

```
Main()
{
int param_1, param_2, param_3;
/*Meghívjuk a subrutine függvényt 3 paraméterrel */
subrutine(param_1,param_2,param_3);
}
```

A subrutine függvény hívásakor a következő történik:

- A jobb szélső paramétertől kezdve betölti a paramétereket a verembe (Legutoljára az első paraméter menti.)
- Menti a visszatérési címet a verembe
- Átadja a vezérlést a függvénynek

Vizsgáljuk meg egy verem képét:



A verem az alacsonyabb memóriacím felé növekszik.
Amennyiben a BP mentése után azonnal betöltjük a veremmutatót(SP) BP-be, utána már szabadon menthetünk bármennyi adatot a stack-re, az már nem befolyásolja a BP-ben elmentett cím és a paraméter távolságát. Azaz az első paraméter mindig azonos ofszetre lesz BP-ben tárolt címtől, MODEL SMALL esetében 4. Mivel távoli (FAR) hívás esetén a CS (kódszegmens) is bekerül a stack-be, ezért itt az ofszet 6 lesz.
A paraméterek által elfoglalt stack nagysága függ a paraméter típusától, amit a szubrutinban is figyelembe kell vennünk. Vagyis a paraméterek címe(az első kivételével) függ attól, hogy milyen típusúak(hány bájtosak).

Mivel MASM generálja- a paraméterátadáshoz szükséges összes – utasítást, a fent említett direktívák használatakor nem kell törődnünk azzal, hogy az utasításokat megfelelő sorrendben írjuk, és azzal sem, hogy a paraméterek melyik veremcímen találhatók.
Híváskor az x és y változókat egészként, paraméter átvételkor pedig word-ként deklaráltuk, ami két-két bájtot jelent. A kurzor címzéséhez pedig egy bájtos adatra van szükség. Így a fordításkor: MOV DH, WORD PTR[PB+4] utasítást kapnánk. Ez azonban az assemblerben nem megengedett. Nem mozgathatunk közvetlen két bájtos adatot egy bájtos helyre. Használnunk kell a BYTE PTR direktívát. Ebben az esetben az utasítás: MOV DH,BYTE PTE WORD PTR[PB +4] lenne. Ezt viszont az assembler nem tudja kezelni. Ezt a problémát úgy tudjuk kiküszöbölni, hogy zárójelezünk. Tehát: MOV DH,BYTE PTE(WORD PTR[PB+4]). Ez pedig visszaírva az eredeti programlistánkba megfelel: MOV DH,BYTE PTE(y) utasításnak. Az assembler ezeket a programozást könnyítő átalakításokat makrók segítségével végzi. Ilyen makró tulajdonképpen az x és y is.

Függvényérték visszaadása: C-ben a függvény érték visszaadásra a következő regisztereket használjuk:

- AL – bájt hosszúságú adatnál,
- AX – szó esetén,
- DX:AX – duplaszavas adatnál.

Szemléltetésül írjunk egy rutint, amely segítségével bekérünk egy karaktert a billentyűzetről.

A C program:

```
Main()
{
clear_screen();
goto_xy(40,25);
```

```
write_string(„Ez egy szöveg”);
while (read_key()!=’ ’);
}
```

Assembler rutin:

```
PUBLIC read_key
Read_key PROC
XOR AH,AH
INT 16h
OR AL,AL
JZ ext_kod
XOR AH,AH
RET
Ext_kod:
MOV AL,AH
MOV AH,1
RET
Read_key ENDP
```

Makrók

Gyakori utasítássorozatok egy általunk kitalált utasítással (macro) kiválthatók. Felhasználásához először deklarálni kell (speciális kezdő- és záró utasítással). A makró törzse tartalmazza az utasításokat. Az első utasításban megadjuk a nevét, esetleges paramétereit. Felhasználásakor elég csak a nevét leírni. A fordítóprogram a makró előfordulásakor behelyettesíti a törzset a makróutasítás helyére.
Assembly felhasználás: name MACRO - utasítások – ENDM

b. Háttértár kezelése.

Egy assembly programban az INT 25h megszakítás segítségével lehet egy lemezről egy blokkot beolvasni. Ez nem karakteresen történik, hanem blokkokban, ehhez csak a forrás és cél címet, valamint a beolvasandó blokkok számát kell megadni. Egy floppy esetében egy blokk mérete 512 byte, így ennyi helyet kell lefoglalni hozzá az adatszegmensben.

```
.MODEL SMALL
Space EQU " "
.STACK
.DATA?
  Block DB 512 DUP(?)
.CODE
main proc
  MOV AX, Dgroup                    ;DS beállítása
  MOV DS,AX
  LEA BX, BLOCK                     ;DS:BX memóriacímre tölti a blokkot
  MOV AL,0                          ;Meghajtó száma
  MOV CX,1                          ;Beolvasott blokkok száma
  MOV DX,0                          ;Lemezolvasás kezdőblokkja
  INT 25h                           ;Beolvasás
  POPF                              ;A veremben tárolt jelzőbitek törlése
  XOR DX,DX                         ;Kiírandó adatok kezdőcíme DS:DX
  CALL write_block
  MOV AH,4Ch                        ;Kilépés
main endp
```

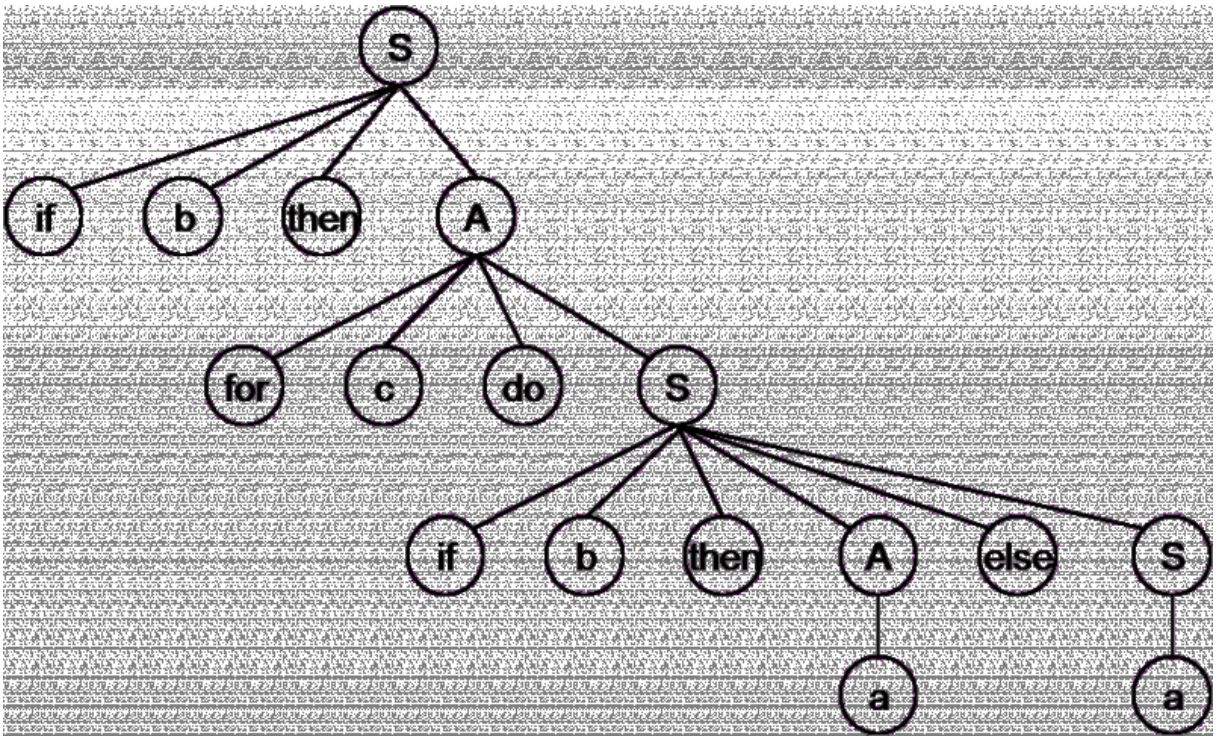
c. Szintaxisfa (levezetési fa)

Legyen G=(N,T,P,S) tetszőleges környezetfüggetlen grammatika. G-beli szintaxis (levezetési) fának nevezünk minden olyan fagráfot, amelynek a csúcsait az N ∪ T halmaz elemei jelölik. Egy mondat szintaxisfája:
A gyökérhez az S kezdőszimbólum tartozik

A levelei a grammatika terminális szimbólumai
A többi pontja a nemterminális szimbólumok halmaza

Egy mondatforma szintaxisfája:
A gyökérhez az S kezdőszimbólum tartozik
A levelei a grammatika terminális és nemterminális szimbólumai
A többi pontja a nemterminális szimbólumok halmaza

A képen látható szintaxisfa a következő környezetfüggetlen grammatikához tartozik: G=[{S,A}, {if, then, else, for, do, a, b, c}, {S→if b then A, S→if b then A else S, S→a, A→ for c do S, A→a}, S].



10.

a. Adatszegmensben megadott szöveg kiírása, képernyő-memória kezelése.

Az adatszegmens használatához deklarálnunk kell azt a programban a .DATA megadásával.

```
.MODEL SMALL
.STACK
.DATA
  adat DB 65
.CODE
main PROC
  MOV AX, Dgroup        ;Adatszegmens helyének lekérdezése
  MOV DS, AX            ;DS beállítása, hogy az adatszegmensre mutasson
  MOV DL, adat          ;Az adat betöltése DL-be
  CALL write_char       ;Karakter kiírása
  MOV AH,4Ch            ;Kilépés
  INT 21h
main endp
```

Mivel előre nem lehet megmondani, hogy a memóriában hova kerül az adatszegmens, ezért szükséges egy DGROUP változó, melynek értékét a betöltő program állítja be. Ezt az értéket kell betölteni a DS regiszterbe. Mivel a memóriából közvetlenül nem lehet adatot tölteni a DS-be, azért közbe kell iktatni az AX regisztert. Adatszegmens használatakor a DS beállítása kötelező.

Képernyő-memória kezelése

A képernyőn megjelenő kép nem más, mint a képernyő-memória leképezése. Ha a memóriába beírunk egy karaktert, akkor az azonnal megjelenik a monitoron. A videó-memória szegmenscíme 0B800h, ez a képernyő első karakterpozíciójának felel meg. Minden karakterpozíciót egy 16 bites szóval jellemezhetünk. Az alsó byte tartalmazza a karakterkódot, a felső a megjelenítő attribútumot.

- Memóriaelem: 0-7 bit karakterkód, 7-15 bit attribútum
- Attribútum: 0-3 bit karakterszín, 4-6 bit háttérszín, 7 bit villogás. A színek RGB szerint tevődnek össze, a 3. bit a világosságbit.

Az attribútum értékének kiszámítása: 128*villogás + 16* háttérszín + karakterszín. Az AX regiszter használata esetén az AL-be kerül a karakterkód és AH-ba az attribútumot.
A képernyőn egy karakter pozíciójának offset címe kiszámítható: 2*((sorszám-1)*sorhossz + karakterpozíció-1). A kettővel szorzás azért kell, mert 2 byte-ot foglal el. Így egy 80 karakter/sor és 50 soros képernyő közepének offset címe 3920.

```
.MODEL SMALL
.STACK
.CODE
main proc
  MOV AX,0B800h              ; Képernyő-memória szegmenscíme ES-be
  MOV ES,AX
  MOV DI,3920          ;A pozíció offset címe
  MOV AL,"*"           ;Kiírandó karakter
  MOV AH, 16*7+4             ;Színkód: szürke háttér, piros karakter
  MOV ES:[DI],AX              ;Karakter beírása a képernyő-memóriába
  MOV AH,4Ch           ;Kilépés
  INT 21h
main endp
END main
```

A képernyő kezdő karaktere a bal felső sarokban van, így az X tengely jobbra, az Y pedig lefelé növekszik. Az első karakter (0,0) offset címe a memóriában viszont 0.

b. A fordítóprogram szerkezete

Fordítóprogram (compiler): olyan számítógépes program, amely valamely programozási nyelven írt programot képes egy másik programozási nyelvre lefordítani. A fordítóprogramok általánosan forrásnyelvi szövegből állítanak elő tárgykódot. A fordítóprogramok feladata, hogy nyelvek közti konverziót hajtsanak végre. A fordítóprogram a forrásprogram beolvasása után elvégzi a lexikális, szintaktikus és szemantikus elemzést, előállítja a szintaxis fát, generálja, majd optimalizálja a tárgykódot.
Compiler feladatai:

- Analízis: a forrásnyelvű program karaktersorozatát részekre bontja, még a szintézis az egyes részeknek megfelelő tárgykódokból építi fel a program teljes tárgykódját.
  - Lexikális elemző (karaktersorozat) (szimbólumsorozat, lexikális hibák): a karaktersorozatban meghatározza az egyes szimbolikus egységeket, a konstansokat, változókat, kulcs szavakat és operátorokat. A karaktersorozatból szimbólumsorozatot készít, ki kell szűrnie a szóköz karaktereket a kommenteket, mivel ezek tárgykódot nem adnak. A magas szintű programnyelvek utasításai általában több sorban írhatók a lexikális elemző feladata, egy több sorba írt utasítás összeállítása is.
    - Szimbólumtábla létrehozása (szimbólum típusa, szimbólum címe)
    - Szóköz karakterek és kommentek kiszűrése
    - Többsoros utasítások összeállítása
  - Szintaktikus elemző (szimbólumsorozat) (szintaktikusan elemzett program, hibák): a program struktúrájának a felismerése. A szintaktikus elemző működésének az eredménye lehet például az elemzett program szintaxisfája vagy ezzel ekvivalens struktúra.
    - Szimbólumok helyének ellenőrzése
    - Ellenőrzi, hogy a szimbólumok sorrendje megfelel-e a programnyelv szabályainak
    - Szintaxisfa előállítása
  - Szemantikus elemző (szintaktikusan elemzett program) (analizált program, hibák): feladata bizonyos szemantikai jellegű tulajdonságok vizsgálata. A szemantikus elemző feladat például az a+b kifejezés elemzésekor az,

hogy az összeadás műveletének a felismerésekor megvizsgálja, hogy az „a” és a „b” változók deklarálva vannak-e, azonos típusúak-e és hogy van-e értékük.

- Konstansok, változók érték- és típusellenőrzése
- Aritmetikai kifejezések ellenőrzése

• Szintézis:

o Kódgenerátor (analizált program) (tárgykód): a tárgykód gépfüggő, generációs rendszertől függő, a leggyakrabban assembly vagy gépi kódú program.

o Kódoptimalizáló (tárgykód) (tárgykód): A kódoptimalizálás a legegyszerűbb esetben a tárgykódban lévő azonos programrészek felfedezését és egy alprogramba való helyezését vagy a hurkok ciklus változásától független részeinek megkeresését és a hurkon kívül való elhelyezését jelenti. Egy jó kódoptimalizálónak jobb és hatékonyabb programot kell előállítania, mint amit egy gyakorlott programozó tud elkészíteni.

c. A programozási nyelvek csoportosítása

Programozás során olyan nyelvet válasszunk, amely tartalmaz olyan konstrukciókat (utasításokat), amely a feladat megfogalmazásában is szerepel.

A programozási nyelvek csoportosítása:

- gépi kód,
- assembly nyelv,
- magasszintű nyelvek

Magasszintű nyelveken belül:

- imperatív nyelvek(PHP,PASCAL): Ezen nyelvek közös jellemzője, hogy a program fejlesztése értékadó utasítások megfelelő sorrendben történő kiadására koncentrálódik. Az értékadó utasítás baloldalán egy változó áll, a jobb oldalán pedig egy megfelelő típusú kifejezés. A szelekció (elágazás) csak azt a célt szolgálja, hogy bizonyos értékadó utasításokat csak adott esetben kell végrehajtani. A ciklusok pedig azért vannak, hogy az értékadó utasításokat többször is végrehajthassunk. Az értékadó utasítások során részeredményeket számolunk ki – végül megkapjuk a keresett végeredményt.
- deklaratív nyelvek,
  - funkcionális nyelvek (Miranda, SML, LISP): A funkcionális nyelveken a kiszámolandó kifejezést adjuk meg, megfelelő mennyiségű bemenő adattal. A programozó munkája a kifejezés kiszámításának leírására szolgál. A program futása közben egyszerűen kiszámítja a szóban forgó kifejezést.

Egy funkcionális nyelvben nincs változó, általában nincs ciklus (helyette rekurzió van). Értékadó utasítás sincs, csak függvény visszatérési értékének megadása létezik. A funkcionális nyelvek tipikus felhasználási területének a természettudományos alkalmazások tekinthetőek.

  - logikai nyelvek (Prolog): Az ilyen jellegű nyelveken tényeket fogalmazunk meg, és logikai állításokat írunk le. A program ezen kívül egyetlen logikai kifejezést is tartalmaz, melynek értékét a programozási nyelv a beépített kiértékelő algoritmusa segítségével, a tények és szabályok figyelembevételével meghatároz.

A logikai nyelvek tipikus felhasználási területe a szakértői rendszerek létrehozásához kapcsolódik.

- objektumelvű nyelvek (java, C++, Delphi): Az OOP nyelveken a program működése egymással kölcsönhatásban álló objektumok működését jelenti. Az objektumok egymás műveleteit aktiválják, melyeket interface-ek írnak le. Ha egy művelet nem végrehajtható, akkor az adott objektum a hívó félnek szabványos módon (kivételkezelés) jelzi a probléma pontos okát.