**1. Informatikai alapfogalmak. A modern számítógép leírása (hardver-, szoftverelemek). Információábrázolás számítógépen (adatok és programok). Az algoritmus fogalma. Elemi algoritmusok (rendezések, keresések). A processzor felépítése és működése. Tárak. I/O rendszer. Perifériák. Operációs rendszerek. Alkalmazói programok. Gépi utasítások osztályozása és végrehajtása. Az assembly forrásprogram felépítése.**

**Modern számítógép leírása**

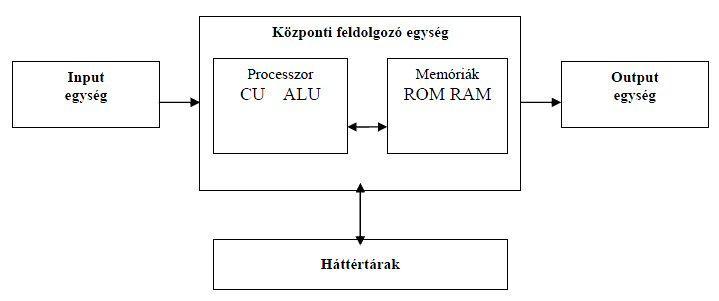
Számítógép eredményes működéséhez számos berendezés, egység összehangolt tevékenységére van szükség. **Hardver:** a számítógép „kézzel fogható részei”, mechanikus és elektronikus alkatrészeinek összessége. **Szoftver:** a számítógép működtetéséhez, feladatok megoldásához használt programok összessége.

Ezek együttes működése biztosítja a számítógépes feladatmegoldás lehetőségét. A számítógép **logikai felépítése:**



Felépítést, működést nagyban meghatározzák Neumann-elvek. Adatok, információk az **input** **egység**en keresztül jutnak a **központi egység**be, aholfeldolgozás történik. **Háttértárak**onadatok és programok tárolása, feldolgozás eredményét **output egység**en keresztül közli.

**A központi feldolgozó egység:**



**Központi feldolgozó egység:** (Central Processing Unit, CPU) két fő alkotóeleme a*processzor*és a *memóriák*, de valójában még tartalmaz ún. **regiszter**eket is (kis tárolókapacitású, de rendkívül gyors elérésű tárolók, gyakran használt adatok, részeredmények tárolására).

*Feladatai*:gép irányítása, a feldolgozási folyamat vezérlése,adatokfeldolgozása, külvilágbólérkező illetve külvilág felé küldött adatforgalom irányítása.

Egyes résztevékenységeket a CPU-n belül önálló részek végzik. Egyik ilyen egység a **processzor**, amely vezérlőegységből (Control Unit, CU) ésaritmetikai-logikai egységből(Arithmetical Logical Unit, ALU) áll + regiszterek. Alkotóelemeit egyetlen áramköri lapon, chipen állítják elő. Az integráltsági fok egy ilyen lapkán óriási, cm2-nként több millió elektronikai alkotóelem.

**Vezérlőegység** feladata a **parancsok** (legkisebb munkafolyamat aprocesszoron belül, lépéseiaz utasítások) feldolgozási sorrendjének meghatározása. Kapcsolatban áll az ALU-val és az operatív memóriával, feladata tehát az ezekkel való kapcsolattartás, valamint a parancsok

1

értelmezése, és végső soron a számítógép egységeinek vezérlése mindezen információk alapján. **ALU** feladata aritmetikai és logikai műveletek végrehajtása. Egyszerű logikai áramkörökből épül fel, melyek összehasonlítani, léptetni, invertálni és összeadni tudnak.

Processzoron belül az információ- és adatcsere **busz**okon (síneken) zajlik. Címbuszon keresztül történik a megfelelő tárolócella kijelölése, az adatbusz adatokat továbbítja, vezérlőbusz pedig vezérlőjeleket továbbít. Folyamatok időbeli ütemezését egy **órajel-generátor** végzi. Az órajel frekvencia a processzor, így a számítógép működési sebességénekjellemző adata.

CPU másik fontos egysége az **operatív tár** vagy memória. Itt tárolódnak az adatok a processzor számára, feldolgozáshoz éppen szükséges adatok (gyorsan állnak rendelkezésre). Alapvetően kétféle típusú memóriát különböztethetünk meg: ROM és RAM.

A **RAM** (Random Acces Memory, közvetlen hozzáférésű memória) olvasható, törölhető, újraírható tár. Áramellátás alatt őrzi meg tartalmát, kikacsoláskor vagy áramszünetkor kiürül. A **ROM** (Read Only Memory, csak olvasható memória) adatai gyártáskor kerülnek fixen a tárba csak olvasható, számítógép felhasználja működéséhez. Tartalma áramellátás nélkül is megmarad, így nem változó adatok, programok kerülnek ide.

**Regiszterek**: processzorhozközeli leggyorsabb tároló, fix hosszúságú (most 32 bites)egységekből áll. Két csoportja: amit felhasználó lát, amit nem lát. **PC** utasításszámláló regiszter tárolja a tárolt program soronkövetkező utasításának memóriacímét. **IR** utasításregiszter tárolja a tárolóból előkeresett, végrehajtandó utasítást, a feldolgozás idejére. Vezérlő- és állapotjelző regiszter: aktuális processzorállapotot visszatükröző, vezérlési előírásokat is tartalmazó regiszter. A táblákba történő íráshoz, vagy olvasáshoz meg kell adni a keresett tárolóhely címét, amit a tárolóhoz tartozó címregiszter (**MAR)** fogad be és ennek tartalma vezérli a memória kiválasztó áramköreit. Az adatok számára a tároló be- és kimenete az adatregiszter (**MDR)**, amely a beírható, vagy kiolvasott adatot ideglegesen befogadja. **AC** az aritmetikai egységakkumulátor regisztere.

Az input és output egységek és háttértárak együtt **perifériák**. Központi egység a perifériákkal az **adatcsatornák**on keresztül tartja a kapcsolatot.

**Perifériák:** CPUés külvilág közötti adatcserét megvalósító eszközök= be-éskiviteli eszközök, csoportjaik adatforgalmi szempontból:

*Input egységek*: adatok, programok, vezérlőjelek ezeken keresztül jutnak a számítógépbe.Leggyakrabban billentyűzet (karakterbevitel) és egér (mutatóeszköz). Scanner, mikrofon, stb. *Output egységek:* információkat, a feldolgozás eredményeit ezekkel jeleníti meg aszámítógép. Leggyakoribb a monitor és a nyomtató, de hangszóró, stb. is ide tartozik.

*Be- és kivitelre egyaránt alkalmas eszközök:* **háttértárak**(adattárolás áram nélkül, hosszútávon). Lényegi jellemzői: tárolótípus, kapacitás, elérési idő. Adathordozó szerint:

Mágneses tárak: adathordozó felületén lévő mágneses réteg alkalmas kétállapotú jelek rögzítésére, adatok rögzítésekor megváltoztathatjuk a felület mágnesezettségét. Olvasáskor a mágneses térben levő vezetőben áram keletkezik, így visszakaphatjuk rögzített jeleket.

Floppy, winchester, mágnesszalag. Optikai tárak: adatokat lézersugár írja egy forgó optikai lemezre (műanyag lemez fémfóliával). Információkat gödröcskék (pitek) tárolják, visszaverődő fényből nyerjük vissza. CD, DVD, Blu-ray.

A számítógép belsejében, (gépház) számítógép egységeit fizikailag összekötő áramköri lap **alaplap**, ezen vannak **processzor**, **memóriák**, **illesztőkártyák**. A perifériák illesztő-

áramkörökkel kapcsolódnak az alaphoz (beépített vagy illesztőkártyák = vezérlőkártyák). A részegységeket **buszrendszer** köti össze, ehhez csatlakoznak illesztőkártyák és beépített illesztők is. A buszrendszer bővíthető.

2

**Szoftverelemek**

A programvezérelt gépek célszerű működését a szoftverek több rétege biztosítja. Aszerint, hogy egy szoftver specifikusan mennyire inkább a gép puszta működtetését, avagy az ember által igényelt feladatmegoldást segíti elő, a következő *funkcionális* csoportokat különböztetjük meg:

* indítóprogram vagy **alapszoftver** – a felhasználó által a legkevésbé manipulálható, a gép üzemszerű működését beállító program(ok), ide tartozik a firmware is;
* **rendszerszoftverek** – a gép és perifériái kommunikációját lebonyolító programok,beleértve a felhasználó oly mértékű kiszolgálását, amely lehetővé teszi a számára más szoftverek elkészítését és üzembe helyezését is;
* **alkalmazói szoftverek** vagy alkalmazások – a felhasználót a számítógép használatántúl mutató céljainak elérésében támogató specifikus programok.

**Rendszerszoftverek:**

* Operációs rendszerek
* Meghajtóprogramok (driverek)
* Segédprogramok

o Fájlkezelők

o Szövegszerkesztők (editorok)

* 1. Tömörítők
* Fejlesztési környezetek
  1. Fordítóprogramok (compilerek)

o Értelmezők (interpreterek) és futtatókörnyezetek o Nyomkövetők és hibakeresők (debuggerek)

o Programszerkesztők (linkerek)

**Alkalmazói szoftverek**

* Irodai szoftverek: Szervezőprogramok, prezentációkészítők, kiadványszerkesztők, táblázatkezelők
* Üzleti alkalmazások: számlázó programok, könyvelő programok, adatbáziskezelők, vállalatirányítási rendszerek
* Grafikai szoftverek: rajzprogramok, képszerkesztők
* Média szoftverek: médialejátszók, médiaszerkesztők
* Kommunikációs szoftverek: levelező programok, csevegő programok
* Hálózati alkalmazások: webböngészők, fájlcserélők
* Játékszoftverek

**Információábrázolás**

*Számítógép minden műveletet számokkal végez***.**Feldolgozás ideje alatt adatok a gépbentárolódnak belső adatábrázolással történik. Adatok belső ábrázolása többféle lehet. Információk **kétállapotú elemek** sorozatával vannak tárolva kettes számrendszer. Használatos még nyolcas (oktális) és tizenhatos (hexadecimális) számrendszerek is. Kettes számrendszerben ábrázolt adatok alapegysége a **bit**, 0 vagy 1 bináris értéket tárolhatja. Nagyon kicsi, ezért egység a **byte** (8 bit).

Bitek helyi értékének figyelembevételével ábrázolunk nagyobb egész számot Adatok tárolására alkalmas legkisebb elérhető (címezhető) tárolóelem a **tárolócella**. A **szó** több ilyen tárolócella összefogása. A címezhető egységek lehetnek byte-ok, de vannak 16 és 32 bites számítógépek is (2 bájt = 16 bit (félszó), 4 bájt = 32 bit (szó), 8 bájt = 64 bit (dupla szó)). Byte többszörösei: 210 byte = 1 Kbyte, 220 byte = 1 Mbyte, 230 byte = 1 Gbyte, 240 byte = 1 Tbyte.

**Előjel nélküli egész számok ábrázolása:** Legtöbb számítógép meghatározott számú bitet(állandó szóhosszúságot) használ az ábrázoláshoz, ha szám ennél rövidebb, nullákkal egészíti

3

ki balról. Az ábrázolás egyenes kódolással történik, azaz az értékek kettes számrendszerbeli alakja felel meg a biteknek. 1 bájton 256 különböző szám, 2 bájton 65536.

**Előjeles egész számok ábrázolása:** pozitív számokat bináris alakjuk ábrázolja, negatívszámokat **előjelbittel** különböztetjük meg, mely 0, ha pozitív számot, 1, ha negatív számot ábrázolunk. Emellett **kettes komplemens** módszerét (nullára történő kiegészítés módszere): kialakításának lépései: szám minden bitjét az ellenkezőjére változtatjuk *egyes komplemens*. Ehhez hozzáadunk 1-et *kettes komplemens*. 1 bájton [-128;127], 2 bájton [-32768, 32767].

**Törtszámok ábrázolása (lebegőpontos):** valós számok ábrázolása, mivel számítógépek csakvéges hosszúságú számjegysorozatokkal képesek dolgozni, a valós számokat egy közelítő értékkel helyettesítve racionális számokká kell alakítani. Bináris lebegőpontos ábrázolás minden számot szorzat alakban ad meg, következő m-*A*k alakban írható fel, ahol m a mantissza, A a számrendszer alapja és k a karakterisztika. Binárisnál m 2k. Mantissza és kitevő is lehet negatív. Előre rögzített, mantissza hány számjegyet tartalmazhat. Mantissza *normált (normalizált)*, ha 1/A**<**m<1 és m***\****0.

Ábrázolás során nem kell tárolni számrendszer alapját, hiszen minden számítás azonos alapú számrendszerben ábrázolt számokkal történik. Meghatározott számú tárolóhelyek mantissza és kitevő számára. *A mantissza* valódi tört, *ábrázolása* történhet kettes komplemens alapján.

*A kitevő ábrázolása* legtöbbször *feszített* módban *(karakterisztika)*, megadja, hogy a mantisszatörtpontját hány helyiértékkel kell áthelyezni. A feszített módban történő ábrázolás számhoz hozzáadunk egy többletet, többlet = 2n-1, ahol n karakterisztika bitszáma. Ezzel negatív kitevő is ábrázolhatóvá válik előjel külön ábrázolása nélkül.

Lebegőpontos ábrázoláshoz gyakran 4 bájtot használnak, de lehet 6, 8, 10 bájtos is. Ábrázolható értéktartomány függ attól, hogy hány biten történik mantissza és karakterisztika ábrázolása. 4 bájtos verzióban 1 előjelbit, 8 bit kitevő, 23 bit mantissza

**Fixpontos ábrázolás:** helyiértékes számrendszerben ábrázolt számjegy-sorozat. Bináris pontfix helyen van (általában az utolsó pozíció utáni helyen), deklarációból derül ki, külön tárolni nem kell. Egyszerű számjegysorozatként tárolja. Legtöbbször egész számok ábrázolására használják, de törtekhez is alkalmas. Előjeles fixpontos esetén első bit előjelbit. Előnye:, műveletvégzés gyorsabb, mint lebegőpontosnál, hátránya: ábrázolási tartomány kisebb.

**Karakterek ábrázolása:** A számítógép által használt teljes jelkészletet alfanumerikusjeleknek, röviden **karakter**eknek nevezzük. Adat lehet betű meg speciális jel is.

**S**zámot az különbözteti meg a számjegyekből álló karaktersorozattól, hogy számokkalaritmetikai műveleteket végzünk, míg karaktereket nem kívánjuk számítási műveletekben felhasználni. Karakterábrázolási módok:

***Kódrendszerek:*** Karakterkódolásnál alapelv, hogy egyeskarakterekhez hozzárendelnek egymeghatározott bitkombinációt. Azt, hogy hány bitből áll egy ilyen kombináció, az határozza meg, hogy a jelkészlet hány karakterből áll (hány eltérő karaktert különböztetünk meg).

***BCD kód*:**Jelkészlet kódolására régen igen elterjedt módszer volt (Binary Coded DecimalCode = binárisan kódolt decimális kód). 6bites kódrendszer 64 karaktert képes ábrázolni. A számjegyek esetén éppen a bináris alak felel meg a kódolt alaknak. Ezt követi az angol ábécé nagybetűi, majd a speciális karakterek 111111-ig. 24 bites szóhosszú gépek alkalmazták. BCD kódban ábrázolt számokkal műveletet végezni nem lehet.

***EBCDIC kód:*** BCDkiterjesztett változata(Extended Binary Coded Decimal Interchange

Code = kiterjesztett BCD kód). Szöveg, grafika, és vezérlőkarakterek ábrázolását teszi lehetővé. Nyolcbites kód (256 lehetséges elem). Nyolc bitet felosztották két négybites hosszúságú részre, első 4 bit a zónarész, második 4 bit pedig a számrész (4 biten egy 16os számrsz. szám ábrázolható). **Zónázott** **(zónás)** **decimális ábrázolás EBCDIC-nél:** minden

4

számjegy ábrázolásakor zónarész F (1111) számjegyek ábrázolása 8 biten, előjel nélküli. Negatív számok előjeleknek megfelelő négybites kombinációk, ábrázolandó szám legalacsonyabb helyiértékű számjegyének megfelelő zónarészben (legkisebb helyiérték utolsó bitje előjel). **Tömörített (pakolt) decimális ábrázolás EBCDIC-nél:** ha csak számokat akarunk ábrázolni, zónarész el is hagyható (úgyis F) bizonyos esetekben 8 biten két számjegy ábrázolható, közel felére csökkenthető lefoglalt bitek száma. Előjeleket nem hagyhatjuk el, utolsó 4 biten jelölni kell az előjelet.

***ASCII*** ***kódrendszer*:**egységesített kódrendszer, karakterekhezbináris kódot rendel.

(American Standard Code for Information Interchange) Eleinte 7 bites kód volt, így 27 = 128 különféle bitsorozat létezett 0-tól 127-ig sorszámozva. Ez alap karakterkészlet (az angol ábécé kis- és nagybetűi, számjegyek, írásjelek). 1 bites kiterjesztéssel +128 karakter 128-255 között kiegészítő karakterkészlet (speciális nemzeti karakterek, ékezetes betűk, vonalrajzoló, görög ABC, stb.) kódlapok, 852-es a magyar karakterkészlet. Szabvány az ASCII karakterkészlet definiálásakor a kódokat két fő csoportba osztotta: grafikus karakterek és vezérlő karakterek csoportjába. *Grafikus karakterek* megjeleníthető, látható, nyomtatható karaktereket értjük, *vezérlő karakterek*, a megjelenítés vezérlésére, formájának kialakítására, valamint az információcsere vezérlésére szolgálnak.

Minden karakterhez egy 0-255 intervallumban lévő számot rendel ASCII kódok, 1 bájtot foglalnak el a memóriában, bájt értéke az ASCII kód, képernyőn a karakter képe jelenik meg.

**Zónázott (zónás) decimális ábrázolás ASCII-nél:** számítógép a kódot hexadecimálisformában rögzíti. 1 bájton ábrázolunk egy számjegyet, első 4 bit a zóna. Előjel nélküli számok ábrázolása nem jelent gondot, a probléma itt is negatív számok, megoldás: a legalacsonyabb helyiértékű számjegy zónarészében jelöljük az előjelet. **Tömörített (pakolt) decimális** **ábrázolás ASCII-nél:** Hasonló megfontolások, mint EBCDIC kódrendszer esetén. Akülönbség annyi, hogy az előjelbitet a legalacsonyabb számjegy zónarészében megtartjuk.

**Az algoritmus fogalma**

Egy adott feladat megoldásának lépéseit, valamint azok sorrendjét az **algoritmus** írja le. **Algoritmus** egyprobléma megoldó lépéssorozatot, mellyel különböző műveletek,átalakítások során eljutunk a megoldáshoz. Az algoritmusban leírt lépéseket a számítógép számára „érthető” formában kell megadnunk: ez a **program**, létrehozásának folyamata pedig a programozás. A program legkisebb funkcionális egysége az **utasítás** (véges számú elemi lépések, amelyeket már nem lehet további részfeladatokká felbontani, az utasítások). A program az utasítások sorozatából áll. Az utasításokat tárolni kell, majd egyenként, a meghatározott (nem feltétlenül lineáris) sorrendben végre kell hajtani. A leírás nyelvét **programozási nyelvnek** nevezzük.

**Elemi algoritmusok (rendezések, keresések)**

A keresések és rendezések valamilyen adathalmazon dolgoznak. Ez egyszerű elemekből áll, és a tagokat meg tudjuk számozni, ezzel megkülönböztetve őket indexelés. Ha nem egyszerű adatokkal dolgozunk, algoritmusok lényegében akkor is ugyanúgy működnek. ***Keresési algoritmusoknál*** az adathalmaz elemein lépkednünk végig (ha valamennyin nem is),ezért a keresési algoritmusokban egy, és csak egy iteráció szerepel.

***Minimum / maximum keresés:*** rendezetlen adathalmazon van értelme.Első lépéskéntelőolvasás: cikluson kívül beolvassuk 1. értéket, ez lesz a kezdő minimum érték. Ezek után végiglépkedünk összes további elemen, vizsgáljuk, hogy kisebb-e a tárolt minimumnál. Ha igen, értékét veszi föl a minimumváltozó (érték indexét is tárolhatjuk). Elemeken végighaladva n – 1 összehasonlítás.

1. *i = 2*
2. *min = K1*

5

1. *Ha i > n akkor menj 7-re*
2. *Ha min > Ki akkor min = Ki*
3. *i = i + 1*
4. *vissza 3-ra*
5. *sikeresen vége*

***Teljes keresés (Szekvenciális keresés):*** konkrét értéket keresünk, rendezetlen adathalmazban.Hasonlít minimum/maximum keresésre, de itt már előfordulhat az is, hogy a keresett elem nincs benne az adathalmazban. Adott a keresett érték, keressük annak az elemnek az indexét, ami egyenlő vele. Minden elemet megnézzük, hogy egyezik-e. Ha igen, sikeresen vége. Sikertelenséget csak akkor jelenthetjük be, ha minden elemet megvizsgáltunk (n db összehasonlítás).

*Strázsa módszer:* adathalmazbautolsó utáni (n + 1) helyre berakjuk a keresett elemet(strázsa) sorozat végének ellenőrzését elhagyhatjuk. Ekkor a ciklus lefutása után vizsgálat: ha sikeresen ért véget a keresés index ≤ n, ha nem index = n + 1 (sikertelenség esetén

|  |  |  |  |
| --- | --- | --- | --- |
| n+1 db összehasonlítás). | |  |  |
| *1. i = 0* | | *1.* | *Kn+1 = K, i = 0* |
| *2. i = i + 1* | | *2. i = i + 1* | |
| *3.* | *Ha Ki = K akkor sikeresen vége.* | *3.* | *Ha Ki ≠ K akkor vissza 2-re* |
| *4.* | *Ha i ≤ n akkor vissza 2-re* | *4.* | *Ha i = n akkor sikertelenül vége* |
| *5. különben sikertelenül vége* | | *5. különben sikeresen vége* | |

***Lineáris keresés (Szekvenciális keresés rendezett sorozatra):*** Rendezett adathalmazonműködik. Végiglépkedünk az elemeken, ciklusban azt vizsgáljuk, hogy az elem kisebb-e keresett elemnél. Ha nem kisebb, akkor vége a ciklusnak. Ciklus utáni vizsgálat elem vagy megegyezett a keresettel (sikeres vége), vagy nagyobb volt nála (sikertelenül vége). Rendezett adathalmazon így előbb megállapíthatjuk, hogy sikertelen volt-e a keresés, mint ahogy azt a teljes kereséssel. Ennél a módszernél is alkalmazható a **strázsa,** végén nem is kell vizsgálni rá, ha keresett elemnél nagyobb értéket adunk neki – azért az ábrázolható tartományba

|  |  |  |  |
| --- | --- | --- | --- |
| beleférjen. | |  |  |
| *1.* | *Kn+1 = K’, i = 0* | *1. Kn+1 = K, i = 0* | |
| *2. i = i + 1* | | *2. i = i + 1* | |
| *3.* | *Ha Ki < K akkor vissza 2-re* | *3.* | *Ha Ki < K akkor vissza 2-re* |
| *4.* | *Ha K = Ki akkor sikeresen vége* | *4.* | *Ha K = Ki és i < n akkor sikeresen vége* |
| *5. különben sikertelenül vége* | | *5. különben sikertelenül vége* | |

***Bináris keresés (Logaritmikus keresés):*** Rendezett adathalmazon keres, +feltétel, hogy azelemeket közvetlenül el tudjuk érni, mivel nem lépkedünk, hanem ugrálunk az adathalmaz elemei között. Keresés során két csoportra bontjuk a sorozatunkat. Kivesszük a középső elemet (ez fogja két részre osztani adathalmazt), és megvizsgáljuk, hogy a keresett elemet választottuk-e ki. Ha igen, akkor a keresésnek sikeresen vége, ha nem, akkor a keresett elem vagy kisebb nála, vagy nagyobb. Ezzel meghatároztuk, hogy melyik halmazban lehet az elem. Elemek felét kidobjuk, ott úgysincs rendre megfelezzük az adathalmazt, (n–1)/2. Ténylegesen nem, csak fölveszünk egy alsó (lower - l) és egy felső (high - h) indexet, és ezek értékeit változtatjuk. Első lépésben l = 1, h = n. Ha a keresett elem kisebb a középsőnél a h = középső–1 lesz, ha nagyobb nála, az l = középső+1.

Sikertelen, ha halmaz elemszáma 1 és az sem egyezik keresettel ~ ha l > h.

|  |  |  |
| --- | --- | --- |
| *1. l = 1, u = n* | *4.* | *Ha K = Ki akkor sikeresen vége* |
| *2. Ha l > u akkor sikertelenül vége* | *5.* | *Ha K < Ki akkor u = i – 1* |
| *3. i = (l + u) / 2* | *6. különben l = i + 1* | |
|  | 6 |  |

*7. vissza 2-re*

*Függvény: Keresés (K[], bal, jobb, K )*

1. *Ha bal > jobb akkor visszaad:–1*
2. *i = (bal + jobb) / 2*
3. *Ha K = Ki akkor visszaad: i*
4. *Ha K < Ki akkor*

*visszaad Keresés(K[], bal, i - 1, K) 5. különben*

*visszaad Keresés(K[], i + 1, jobb, K)*

***Rendezések*** alapja: elemeketösszehasonlítjuk, ha kell,megcseréljük. Első lépésben nemtudjuk az összes elem helyét megkerestetni, ezért minden rendezésre jellemző, hogy két ciklust, egy belső és egy külső ciklust tartalmaz. Külső ciklusban általában értékadások történnek, kilépési feltétele határozza meg, hogy a rendezés mikor fejeződik be. A belső ciklusban az elemeken vagy még inkább az elemek egy részhalmazán lépkedünk végig. Ez a rész tartalmazza az összehasonlítást. Elemek cseréje rendezési módszertől függően vagy a belső vagy a külső ciklusban szerepel.

***Kiválasztásos rendezés:*** minimum/maximumkereséssel van kapcsolatban. Külső ciklus aztadja meg, hogy az adathalmaz melyik helyére keressük a megfelelő elemet. A belső ciklusban megkeressük a legkisebb elemet, és amikor befejeződik berakjuk a megfelelő helyre. Elemek cseréje külső ciklusban. Külső ciklus n-1-szer ismétlődik (utolsó helyre nem kell keresni). Belső ciklus kezdőértéke a külső ciklus aktuális értéke rendezett részen már nem keresünk. Csökkenő sorrendbe rendezésnél maximum értéket keresünk, ha visszafelé rendezzük az adatokat, először legutolsó elemet határozzuk meg, külső ciklus visszafelé halad. Egyenletesen rossz teljesítményű .

1. *i = 1*
2. *min = i, j = i*
3. *j = j + 1*
4. *Ha Aj < Amin akkor min = j*
5. *Ha j < n akkor vissza 3-ra*
6. *t = Ai Ai = Amin Amin = t*
7. *i = i + 1*
8. *Ha i < n akkor vissza 2-re*
9. *vége*

***Beszúrásos rendezés:*** elemhezkeressük a helyet, lineáris kereséssel van kapcsolatban. Sorravesszük az elemeket, és a már rendezett részhalmazban egkeressük a helyét. A külső ciklus 2-től n-ig halad. A belső ciklusban a rendezett adathalmaz elemein végighaladva minden olyan tagot átlépünk, ami nagyobb nála. A belső ciklus meghatározza az elem helyét. Fennmaradó elemeket eltoljuk (helyet kell csinálni beszúrandó elemnek).

1. *i = 1*
2. *i = i + 1*
3. *j = i – 1, x = Ai*
4. *Ha Aj ≤ x akkor menj 8-ra*
5. *Aj+1 = Aj*
6. *j = j – 1*
7. *Ha j > 0 akkor vissza 4-re*
8. *Aj+1 = x*
9. *Ha i ≠ n akkor vissza 2-re*
10. *vége*

***Buborékrendezés:*** előző két rendezésnél egy-egy elem rendezésekor sok mellékes cseretörténik, ez ezen próbál változtatni. Minimum kiválasztással áll kapcsolatban. Vesszük az első

7

elemet, és az mondjuk, hogy ez a legnagyobb. Ez után sorban végiglépkedünk az adathalmaz elemein, és ha ennél az elemnél kisebbet találunk, akkor megcseréljük őket, ha nagyobbat, akkor ettől kezdve az lesz a legnagyobb elem. Így járva el, a legnagyobb elem a ciklus végén a helyére kerül. Egy cserével adott elem mindig közelebb kerül a helyéhez. Az algoritmus futási ideje javítható, ha a már rendezett elemekre nem hajtjuk végre az összehasonlítást. Ekkor elegendő, ha a belső ciklus (n–k+1)-ig (k a külső ciklus aktuális értéke) hajtódik végre. Hatékonyságán azzal is javíthatunk, ha alkalmazunk egy *voltcsere* változót belső ciklus végrehajtása előtt meghatározott értéket kap, amikor cserét végzünk, megváltoztatjuk. A külső ciklus kilépési feltétele pedig az, amikor a *voltcsere* értéke az eredeti maradt.

Néha kifejezetten lassú, mert egy-egy elem csak közelebb kerül a helyéhez, és nem a valódi helyét foglalja el (kivéve az aktuális legnagyobb/legkisebb elemet) felesleges cserék. Ha már rendezett sorozatot kell rendezni, a buborékrendezés (*voltcsere* változóval) minden más rendezéshez képest előbb végez a sorozat ellenőrzésével.

|  |  |  |
| --- | --- | --- |
| *1.* | *[ Ciklus i = (n-1), …, 1-re ] hajtsuk* | *1. voltcsere = 1, i = 1* |
| *végre 2-t* | | *2. Amíg voltcsere = 1 és i ≤ n hajtsuk végre* |
| *2.* | *[ Ciklus j = 1, …, i-re ] hajtsuk végre 3-t* | *3, 4-et* |
| *3.* | *Ha Aj > Aj+1 akkor* | *3. i = i + 1, voltcsere = 0* |
| *cseréljük meg Aj Aj+1* | | *4. j = n, n-1, …, i-re hajtsuk végre 5-t* |
|  |  | *5. Ha Aj < Aj-1 akkor* |
|  |  | *t = Aj , Aj = Aj-1 , Aj-1 = t , voltcsere = 1* |

***Gyorsrendezés (Quick sort):*** halmazt felosztjuk,rendezni kívánt sorozat két részhalmazaközötti eltéréseket hozza rendbe. A belső ciklus lefutásakor a két részhalmaz elemei általában továbbra is rendezetlenek, de rajtuk is elvégezve a rendezést - végül eljutva az egyelemű halmazokig - a teljes sorozat rendezett lesz. Gyorsaságát két dolognak köszönheti. Jó esetben a rendezés külső ciklusa log2 n-szer hajtódik végre, és egy csere elvégzésével két elem kerül a neki megfelelő halmazba. A külső ciklus csak akkor hajtódik végre log2 n-szer, ha a kitüntetett elem rendre mindig a sorozat középső eleme, ez pedig csak speciális esetben fordul elő, ezért a külső ciklus ennél némiképp többször fut le, legrosszabb esetben n-szer.

Az algoritmus megvalósításakor a problémát az jelenti, hogy nem tudjuk, hogy hol van a két részhalmaz határa. (Nem tudjuk, hogy hol van a középső elem, hiszen a sorozat rendezetlen.) Ezért általában úgy valósítják meg a gyorsrendezést, hogy kivesznek egy elemet az adathalmazból, és ehhez hasonlítják a többi elemet. Azaz megkeressük a kiválasztott elem helyét.

* 1. *Ha a sorozat hossza ≤1, akkor vége*
  2. *legyen X tetszőlegesen választott elem*
  3. *helyezzük el X elé az X-nél kisebb elemeket; X után az X-nél nagyobb elemeket.*
  4. *gyorsrendezés az X-nél kisebb részsorozatra*
  5. *gyorsrendezés az X-nél nagyobb részsorozatra*
  6. *vége*

1. *lépés kifejtve:* 
   * 1. *l=1, u=n, X=A [e+u/2]*
     2. *Amíg l ≤u hajtsuk végre 3,5,7*
     3. *Amíg Ai<X hajtsuk végre 4-t*
     4. *l=l+1*
     5. *Amíg Au >X hajtsuk végre 6-t*
     6. *u=u-1*
     7. *Ha l≤u akkor cseréljük meg Al,Au elemeket és l=l+1, u=u-1*

8

***Összeválogatás:***

1. *A sorozatot 2 részre vágjuk (b,c)*
2. *A b és c sorozatból állítsunk elő összefésüléssel egy olyan sorozatot, ahol a szomszédos elem párok rendezettek*
3. *Az 1,2-est ismételjük, de mostmár az elem négyesek lesznek rendezettek.*
4. *ismételjük meg az előzőt, de mostmár az elem négyesek lesznek rendezettek.*
5. *mindaddig folytatjuk, amíg az egész sorozat rendezett lesz.*

B1,b2,…,bq c1,c2,…,cr a

1. *i=1, j=1, k=1*
2. *amíg i≤q és j≤r hajtsuk végre 3-t*
3. *ha bi<cj akkor ak=bi, i=i+1, k=k+1; különben ak=cj, j=j+1, k=k+1*
4. *ha i≤q akkor b sorozat végének másolása a-ba*
5. *ha j≤r akkor c sorozat végének másolása a-ba*

**Operációs rendszer:** *Számítógépes rendszer felépítése*:alkalmazási rendszer –nyelvirendszer – operációs rendszer réteg – hardver réteg.

Az operációs rendszer egy olyan programrendszer, amely a számítógépes rendszerben a programok végrehajtását végzi, ütemezi a programokat, elosztja az erőforrásokat és biztosítja a számítógép és a felhasználó közötti kommunikációt. **Erőforrás szemléletben:** A folyamatok egy olyan csoportja, amely a felhasználói folyamatok között elosztja az erőforrásokat. **Felhasználói szemléletben:** A folyamatok egy olyan csoportja, amely megkíméli a felhasználókat a hardverkezelés nehézségeitől és kellemesebb alkalmazói környezetet biztosít. **Fő funkciói:** Rendszeradminisztráció, programfejlesztési támogatás, alkalmazói támogatás. Következő funkciókat kell ellátnia:

1. **Eszközkezelők (Device Driver):** Felhasználók elől el fedik a perifériák különbségeit,egységes kezelői felületet kell biztosítani.
2. **Megszakítás kezelés (Interrupt Handling):** Alkalmas perifériák felől érkező kiszolgálásiigények fogadására, megfelelő ellátására.
3. **Rendszerhívás, válasz (System Call, Reply):** azoperációs rendszer magjának ki kellszolgálnia a felhasználói alkalmazások (programok) erőforrások iránti igényeit úgy, hogy azok lehetőleg észre se vegyék azt, hogy nem közvetlenül használják a perifériákat programok által kiadott rendszerhívások, melyekre rendszermag válaszokat küldhet.
4. **Erőforrás kezelés (Resource Management):** Az egyes eszközök közös használatábólszármazó konfliktusokat meg kell előznie, vagy bekövetkezésük esetén fel kell oldania.
5. **Processzor ütemezés (CPU Scheduling):** Az operációs rendszerek ütemező funkciójánaka várakozó munkák között valamilyen stratégia alapján el kell osztani a processzor idejét, illetve vezérelnie kell a munkák közötti átkapcsolási folyamatot.
6. **Memóriakezelés (Memory Management):** Gazdálkodnia kell a memóriával, fel kellosztania azt a munkák között úgy, hogy azok egymást se zavarhassák, és az operációs renszerben se tegyenek kárt.
7. **Állomány- és lemezkezelés (File and Disk Management):** Rendet kell tartania ahosszabb távra megőrzendő állományok között.
8. **Felhasználói felület (User Interface):** A parancsnyelveket feldolgozó monito utódja,fejlettebb változata, melynek segítségével a felhasználó közölni tudja a rendszermaggal kívánságait, illetve annk állapotáról információt szerezhet.

***Az operációs rendszerek csoportosítása:***

9

Az operációs rendszer működteti a hardvert, az erőforrások optimális kihasználására törekszik, kapcsolatot tart a gép különböző részei között, kiszolgálja a felhasználót.

1. *Egy felhasználós*: tárban egyszerre csak 1 program működik, a programok egymás utánfutnak – kötegelt (batch) feldolgozás.
2. *SPOOLING üzemmód*:feldolgozásnem sorban egymás után,felhasználó tetszőlegessorrendben adja meg a parancsokat. Ezt op.rendszer a háttértárolón tárolja, és ha van ideje, végrehajtja, és eredményt ismét a háttértárolóra teszi, amit felhasználó bármikor megtekinthet.
3. *Többfelhasználós, valós idejű (real time)*: az adatokat azonnal feldolgozza.
4. *Információs rendszer üzemmód*: központi gépre csatlakoznak a terminálok, amiken ugyanaza program fut, ugyanazok a fájlok működnek interaktívan (oda-vissza irányban).
5. *Időosztásos (time shearing)*: központi gépen, terminálokon keresztül több felhasználódolgozik egyidőben, más-más programokkal és fájlokkal. Op.rendszer felváltva ad időt a felhasználóknak (virtuális memóriát használ).

**DOS**: egyfelhasználós, egyedi programfuttatású, interaktív operációs rendszer, batch és realtime feldolgozás. **WINDOWS**: 3.x-ig nem önálló, DOS-ra épül. 95/98 már telepíti magával a DOS-t. Grafikus felületű, multitasking üzemmódú (egyszerre több program fut időosztásosan). **UNIX**: többfelhasználós, multitaskingos rendszer. **WINDOWS NT**: jhálózati

WINDOWS. **NOVELL**: DOS-ra épülő hálózatos rendszer. **OS/2**: grafikus, multitaskingos operációs rendszer.

**Alkalmazói programok**

Az operációs rendszer és a felhasználói programok között az a határozott különbség, hogy amíg az operációs rendszer, a hardvert működtető rendszer, addig a **felhasználói programok** általános igényeket kielégítő, széles körben használható, adott feladatok megoldására, statisztikai, ill. egyéb adatok feldolgozására kifejlesztett, az operációs rendszerre épülő programok, programrendszerek.

A programokat - feladataiktól függően - az alábbiak szerint lehet csoportosítani:

* operációs rendszerek,
* felhasználói programok.

Az *operációs rendszer* teszi lehetővé a számítógép elemi szintű kezelését, másrészt vezérlik a felhasználói programok végrehajtását, mintegy hidat teremtve a futó program és a gép eszközkészlete között.

A felhasználói programok a számítógépes feldolgozás eszközei, segítségükkel tudunk valamilyen konkrét feadatot elvégezni a gépen. A feladatok jellegétól függően ezek a programok lehetnek:

* programnyelvek (pl. Pascal, C, Basic),
* segédprogramok (pl. fájlkezelők, tömörítők, víruskeresők),
* irodai programcsomagok (pl. szövegszerkesztők, táblázatkezelők, adatbáziskezelők),
* …

**Gépi utasítások osztályozása és végrehajtása**

Gépi kódú utasítások matematikai műveletekhez hasonlóan épülnek föl. Megadjuk, hogy melyik műveletet végezzük el és milyen adatokon. Megadásuk prefix. Művelet alatt nem csak aritmetikai/logikai műveletet lehet érteni, hanem más, számítástechnikai utasítást is, például egy tárrészbe érték btöltése. Utasítás meghatározza, hogy hány operandust követel meg. Gépi szinten csak egyszerű aritmetikai és logikai műveletek vannak. Az utasításokat különböző bájtértékek, számok jelzik. Ezen utasítások megadási módjai:

**Négycímes utasítás:**

*utasítás kód | első operandus | második operandus | eredmény címe | következő utasítás*

10

Utasítás megadása után megadjuk 2 operandust + tárcímet ahova eredmény megy. Ezután program következő utasításának tárcíme. Már nem használják.

**Háromcímes utasítás:** *utasítás kód | első operandus | második operandus | eredmény címe*

Program utasításait tárban folytatólagosan helyezzük el ismerve adott utasítás hosszát és címét, ki tudjuk számoltatni a következő utasítás címét, nem kell megadi (automatikusan hardver végzi). Program Counter (PC) nevű regiszter következő utasítás memóriacíme. Minden utasítás végrehajtásakor automatikusan kiszámolja következő utasítás címét és megváltoztatja PC értékét, ezzel követve az utasítások sorrendjét. Mai gépek e szerint az elv szerint működnek.

**Kétcímes utasítás:** *utasítás kód | első operandus | második operandus*

3. címet is elhagyhatjuk, ha az eredmény címe 1. vagy 2. operandus helyére töltődik (utasítás kódja adja meg, hogy az adott utasítás, hogyan működik.) Operandus eredeti értékét elveszítjük természetesen.

**Egycímes utasítás:** *utasítás kód | első operandus*

Egyik operandus az Akkumulátorban helyezkedik el, eredmény is ide kerül, művelet megadásakor csak a másik paramétert adjuk meg. Végrehajtása előtt az első operandust az Akkumulátorba kell töltenünk, és művelet(ek) befejezése után, ha szükségünk van az eredményre, ki kell vennünk az Akkumulátorból. Akkor van értelme, ha eredménnyel folyamatosan, több műveletet végzünk (regiszter elérése sokkal gyorsabb, mint operatívtáré). Nem minden utasítás igényel két operandust, vannak egyoperandusú és operandus nélküli utasítások is, továbbá nem minden utasítás ad vissza értéket ugró (JUMP) utasítás, üres utasítás (NOP).

**Címzési módok:** operandusok megadása különböző módokon történhet, attól függően,hogyaz adatott hogyan érjük el, különböző címzési módokat különböztetnek meg.

**Közvetlen adat:** *utasítás | adat*.Ekkor operandus helyén maga a feldolgozandó adat szerepel.Paraméter egy konstans, állandó érték.

**Direkt (közvetlen) címzés:** *utasítás | címhivatkozás*cím[adat]. Operandus helyén azoperatív memória egy tárcíme van, ezen a címen található az adat.

**Regiszteres direkt címzés:** *utasítás | regiszterhivatkozás*regiszter[adat]. Operandushelyén egy regiszterhivatkozás szerepel. Ebben a regiszterben található az adat.

**Indirekt címzés:** *utasítás | címhivatkozás*cím[címhivatkozás]cím[adat]. Operandushelyén egy cím, operatívtár egy másik címére mutat, ez megadja, hogy hol van tárban az adat.

**Regiszteres indirekt címzés:** *utasítás | regiszterhivatkozás*regiszter[címhivatkozás]cím[adat]. Operandus helyén egy regiszterhivatkozás, regiszter egy címhivatkozást tartalmaz, az adat ezen a tárcímen található.

Következő két címzési mód nagyobb adathalmazt kezel, ennek minden egyes eleme azonos nagyságú tárterületet foglal el (pl. 1 bájt), és a memóriában folytatólagosan helyezkednek el. Az első elem kezdőcíme megadja a bázist, azt a memóriacímet, ahol az adathalmaz kezdődik. Ettől kezdve adatok elérése indexeléssel (első elem indexe nulla, *n.* elem tárcíme: b + (i\*l), ahol b báziscím (base pointer); l adatelemek mérete (length); i adott elem indexe. Az első elem címe = báziscím.

**Indexregiszteres címzés:** *utasítás | regiszterhivatkozás*regiszter[index]. Adatok hosszánkívül báziscím adott, előre meghatározott. Az index értékét egy regiszterben tároljuk. Regiszter értékét változtatva adott memóriaterület minden egyes elemén végig tudunk lépkedni. Így is fel lehetne írni: *utasítás | regiszterhivatkozás* regiszter[index] függvény(cím) cím[adat]. Függvény értékét a számítógép a címzési mód alapján automatikusan számolja ki, nem nekünk kell vele foglalkoznunk.

11

**Bázisregiszteres címzés:** *utasítás | regiszterhivatkozás*regiszter[bázis]. Adatok hossza ésaz index adott. Egy regiszterben a báziscímet tároljuk. Ugyanúgy működik, mint az indexregiszteres, csak itt a regiszter értékét változtatva a báziscímet változtatjuk meg, így különböző memória területeken elhelyezkedő adatok ugyanazon sorszámú elemét érjük el. A két címzés keverhető, datok változtatásával elérhető, hogy az indexregiszteres címzés úgy működjön, mint a bázisregiszteres, és fordítva.

**Gépi kódú utasítások feldolgozásának lépései:**

1. PC alapján feldolgozandó utasítás kikeresése a tárban, átvitele az utasításregiszterbe.
2. PC++, hogy soronkövetkező utasítás helyét adja meg a tárolóban.
3. Lehívott utasítás értelmezése, dekódolása mit végrehajtania az utasítás hatására.
4. Szükséges adat előkészítése művelet elvégzéséhez, cím alapján átviszi az utasítás által meghatározott helyre, (legtöbbször AC).
5. Az előírt művelet végrehajtása.
6. Eredmény (ha van) előírt helyre való helyezése, vissza első lépéshez, és újrakezdés.

**Az assembly forrásprogram felépítése**

Az assembly nyelv közvetlenül a processzor gépi kódjához kapcsolódik, az assembly nyelv utasításai és a processzor utasításai között egyértelmű megfeleltetés van. Ilyen módon a programozó a processzor minden utasítását elérheti, de ez azt is jelenti, hogy egy adott assembly nyelven csak **egy adott processzorhoz** lehet programot írni, hiszen a különböző típusú processzoroknak eltérő utasításkészlete van. Minden processzortípusnak saját assembly nyelve van és az egyiken megírt programot más processzorok nem tudnak futtatni. Assembly nyelven a programozók **mnemonikokat** használnak a processzor utasításainak azonosítására.

**Általános** rendeltetésű regiszterek az **AX**, **BX**, **CX**, **DX** neveket viseli.

Egy ilyen regiszter nem más mint egy 16 bites adat tárolására alkalmas rekesz, ahová 0 - 65535-ig bármilyen számot írhatunk. Használhatjuk 16 ill. 2\*8 bitesként is, mert egy ilyen rekesznek van alsó ill. felső része. Pl.: Az AX regiszternél **AL** (alsó) ill. **AH** (felső) része.

**Szegmens** regiszterek:

**CS** (Code Segment) kódszegmens

**DS** (Data Segment) adatszegmens **ES** (Extra Segment) extraszegmens **SS** (Stack Segment) veremszegmens

**Index**regiszterek:



**SI** (Source Index) forrásindex **DI** (Destination Index) célindex

**További két** 16 bites regiszter:



**BP** (Base Pointer) bázismutató

**Flag** regiszter

**A program felépítésének szabályai az .EXE programnál:**

Kód **Segment**

**assume** CS:Kód, DS:Adat, SS:Stack

Start: **.**

**.**

**.**

Kód **Ends**

Adat **Segment**

12

**.**

**.**

**.**

Adat **Ends**

Stack **Segment**

**.**

**.**

**.**

Stack **Ends**

**End** Start

**A program felépítésének szabályai a .COM programnál:**

Kód **Segment**

**assume** CS:Kód, DS:Kód

**Org** 100h

Start: **.**

**.**

**.**

Kód **Ends**

**End** Start

Az .EXE és a .COM program között a különbség, hogy .EXE bármilyen hosszú lehet, míg a

.COM-nak bele kell férnie egy szegmensbe, azaz nem lehet 64 Kbyte-nál nagyobb. A **Segment** jelöli a szegmens kezdetét, melynek a neve az előtte álló címke, ami bármi lehet.Aszegmens végét az **Ends** jelzi. Az **assume** szerepe, hogy a szegmensregiszterekbe a hozzá tartozó szegmenscímet töltse. Az .EXE programoknál az assume a **DS** regiszterbe nem az értéket tölti, amit később használni szeretnénk, ezért a programunkba ezt külön be kell állítani. Az **SS** regiszternek nem kötelező értéket adni tehát a címkét is elhagyhatjuk, és az assume sorból is törölhetjük. Ahogy a .COM program felépítésénél látjuk, ezenkívül a **CS** értéke megegyezhet **DS** értékével, ilyenkor ugyanaz lesz a kód illetve az adatszegmensünk. Az **Org** meghatározza a program kezdőcímét a szegmensen belül, ezt célszerű 100h-nak választani, mert az ez alatti memóriaterületen van az operációs rendszer programunkra vonatkozó paraméterei, és adatai.

A most használatos magas szintű programnyelvek komplex szerkesztési rendszerrel működnek, azaz egy menürendszer segítségével eljuthatunk a program begépelésétől a futtatható programig. Assembly típusú programoknál általában ezeket a lépéseket külön kell választani. Nézzük végig és jellemzzük a végrehajtás sorrendjében ezeket a lépéseket.

1. ***A program forráskódjának létrehozása.*** Ezt egy tetszőleges olyan editorral begépelhetjük,mely text formában, minden kísérő jeltől mentesen hozza létre a forrás szöveget. Kiterjesztésként célszerűen az ASM-et szokás adni, amiről felismerhető lenne az adott szövegfájl típusa.
2. ***A forráskód szintaktikus elemzése és egy félig lefordított forma előállítása.*** Ebben afázisban a fordítóprogram veszi át a szerepet (MASM, TASM, …). Ez formailag ellenőrzi

13

a begépelt szöveget és ha hibátlan létrehoz egy félig lefordított formát, ami nem tartalmazza a külső hivatkozásokat, valamint az előre hivatkozások pontos értékeit. Az így keletkezett fiájl kiterjesztése OBJ lesz, ami a félig lefordított -object- forma első három betűje. A fordító program igény szerint készít egy fordítási listát és egy úgynevezett keresztreferencia táblát is, ami a használt adatok hivatkozásait tartalmazza. A következő listát a fordítóprogram által készítette. Jellemzőit az előző fejezetben elemezhetjük.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 0000 | |  | adat segment | ;az adatszegmens kezdete |
| 2 | 0000 000C | | valami dw 12 | ;valami:=12 |
| 3 | 0002 44 4F 4C 47 4F 5A 41 | | szoveg db 'DOLGOZAT ! ' | ;karaktersorozat definíciója |
| 4 |  | 54 20 21 20 |  |  |
| 5 | = 000B | | hossz equ ($-szoveg) | ;hossz:=az előző |
|  |  |  |  | ;karaktersorozat hossza(9) |
| 6 | 000D |  | adat ends | ;az adat szegmens vége |
| 7 | 0000 |  | program segment | ;a program szegmens |
|  |  |  |  | ;kezdete |
| 8 |  |  | assume cs:program, ds:adat | ;szegmens regiszterek |
|  |  |  |  | ;szegmens való rendelése |
| 9 |  |  |  |  |
| 10 0000 | |  | start: | ;az első utasítás címkéje |
| 11 0000 | | B9 0000 | mov cx,0 | ;cx:=0 |
| 12 0003 | | BA 184F | mov dx,184fh | ;dx:=184fh |
| 13 0006 | | B4 4C | mov ah,4ch | ;visszatérés a DOS-ba |
| 14 0008 | | CD 21 | int 21h |  |
| 15 000A | |  | program ends |  |
| 16 | |  | end start |  |

A lista fájlban található még a program szereplő szegmensek listája is. Először a szegmens neve olvasható, majd a hossza hexadecimálisan, valamint az úgynevezett *igazítás, a* *betöltés milyensége,* illetve *a szegmens* típusa található. (Ezeket a szegmens definíciójánálelemezzük majd.)

|  |  |  |  |
| --- | --- | --- | --- |
| Segments and Groups: | |  |  |
|  | N a m e | Length Align Combine Class | |
| ADAT . . | . . . . . . . | 000D | PARA NONE |
| PROGRAM . . . .000A | |  | PARA NONE |

Ebben a listában találhatóak a használt szimbólumok is. Először a szimbólum nevét látjuk, utána a szimbólum tartalmának ’típusa’, az értéke, a típusa. Így a HOSSZ egy számot tartalmazó konstans, a start egy címke (Label) aminek offsetje 0 és a PROGRAM szegmensben található. A SZOVEG egy karaktersorozat az ADAT szegmensben melynek offsetje 2. A VALAMI egy kétbájtnyi adat ami az ADAT szegmensben található offsetje 0. Végül a forrásfiájl neve található amiből ez a lista file készült.

|  |  |  |
| --- | --- | --- |
| Symbols: |  |  |
| N a m e | Type | Value Attr |
| HOSSZ . . . . . . . . .NUMBER | | 000B |
| START . . . . . . . . . L NEAR 0000 | | PROGRAM |
| SZOVEG . . . . . . . . L BYTE 0002 | | ADAT |
| VALAMI . . . . . . . . L WORD | | 0000 ADAT |

@FILENAME . . . . . . . . . . . TEXT ass1

A keresztreferencia fájl nem olvasható bináris fájl.

1. ***A program szerkesztése, a nem definiált címek előállítása, a futtatható program elkészítése.*** Ezt a tevékenységet ’*linkelésnek’*szokták nevezni. Általában egy LINK nevű

14

program végzi ezt a funkciót. Eredményül egy EXE vagy egy COM kiterjesztésű fájlt kapunk.

1. ***A program betöltése, futtatása.*** Parancs üzemmódban leírjuk a futtatandó program nevét,ezáltal betöltődik a program, és a kódszegmens első végrehajtható utasításának végrehajtásával elkezdődik a program működése.

Egy teljes assembly program legalább 1 legfeljebb 4 szegmensből állhat. Mindenképpen lennie kell egy kódszegmensnek, hiszen a program betöltésekor csak a kódszegmens regiszter kap értéket. Minden szegmenshez hozzárendelünk egy szegmens regisztert, ami mint az előző fejezetben láttuk, az abszolút cím képzésében játszik fontos szerepet. A szegmensek és a jellemzően hozzájuk rendelt szegmensregiszterek jellemzése:

* ***Kódszegmens.*** Jellemzően a program végrehajtható utasításaittartalmazza. Ha csak ez azegyetlen szegmensünk van, akkor ez tartalmazhatja a szimbolikus memória címeket, és azok jellemzőit is. Definiálhatunk itt eljárásokat és makrókat. A hozzárendelt szegmensregiszter a CS, ami program betöltésekor kap értéket.
* ***Adatszegmens.*** Jellemzően a programban használt adatok –amik lényegében szimbolikusmemória címek- leírására szolgál. A szokásosan hozzárendelt szegmens regiszter a DS. Ezenkívül hozzárendelhetjük az ES és CS regisztereket is.
* ***Extraszegmens.*** Olyan szegmens, amit általában valamilyen már meglévő adatradefiniálnak rá. Ilyen lehet például a karakteres, vagy a grafikus képernyő. Jellemzően hozzárendelt szegmensregiszter.
* ***Stackszegmens, vagy verem szegmens.*** Alapvetően az eljárások végrehajtásában játszanakfontos szerepet. Az eljárás meghívásakor ide kerül a visszatérési cím. Ide helyezhetjük az eljárások paramétereit, valamint olyan adatokat, amiket átmenetileg meg kell őriznünk. A bp és az sp regiszterekkel címezhetjük. Maga a szegmens tulajdonképpen egy sok bájtból álló tömb, amit két művelettel használhatunk: csak a tetejére rakhatunk adatot, illetve csak a tetejéről vehetünk ki adatot. Mérete valamilyen 2 hatvány, általában 256, vagy 4096 bájt, de tetszőleges érték lehet.

Nézzük most meg a szegmens definíciók szintaktikus, formai szabályoknak megfelelő leírását. Általános alakja:

***NEVE*** ***SEGMENT*** IGAZÍTÁS KOMBINÁCIÓ OSZTÁLY

;A szegmens elemei

***NEVE ENDS***

A félkövér-dölt betűtípussal szedett mezők kötelezőek, a továbbiaknak van feltételezett értékük, így ha nem definiáljuk, akkor az úgynevezett *default* érték lesz az aktuális. Vegyük sorba az egyes mezőket:

***NEVE:*** a szegmens azonosítója. A szokásos szimbólumképző karaktereket tartalmazhatja,betűk, számjegyek, aláhúzás karakter.

***SEGMENT*** :alapszó, a szegmens definíció kezdetét jelöli.

IGAZÍTÁS: a szegmens első bejegyzése abszolút címének jellemzője. Lehetséges értékei: BYTE: ebben az esetben az adott szegmens közvetlenül az előzőleg definiált szegmens után kezdődik.

WORD: a szegmens páros című bájton kezdődik.

PARA: a szegmens paragrafus határon kezdődik. Egy *paragrafus* 16 bájt hosszú, így ezek a szegmensek 16-al osztható címen kezdődnek.

AT abszolút cím: a szegmens az abszolút címnek megfelelő helyre definiálódik, ami lényegében az adott terület szimbolikus névvel való azonosítását teszi lehetőveé. Példaként tekintsük az AT 0B800H igazítást, ami a karakteres képernyő ’újra definiálását’ jelenti.

Feltételezett értéke: PARA.

15

KOMBNÁCIÓ: az adott szegmens memóriabeli elhelyezkedését adja meg. Lehetséges értékei:

COMMON: az ilyen bejegyzésű szegmensek közül csak az utoljára használt van a memóriában. Az azonos osztályhoz tartozó szegmensek -például több különböző adat szegmens használata esetén- ugyanarra a memória területre töltődnek, aminek a hossza a legnagyobb ilyen típusú szegmens hosszával egyezik meg.

PUBLIC: az azonos osztályba tartozó szegmensek közül mindegyik használt a memóriában van. Így ha több adatszegmensünk van, célszerűbb az előző bejegyzést használni. Feltételezett értéke: PUBLIC.

OSZTÁLY: a szegmens típusára utaló szöveg konstans. A ’CODE’ bejegyzés a kód szegmensre, a ’DATA’ az adat szegmensre, míg a ’STACK’ a verem szegmensre utal.

16