Assembly programozás levelező tagozat

Németh Gábor

Szegedi Tudományegyetem
Képfeldolgozás és Számítógépes Grafika Tanszék

2011-2012-2

Tematika

- Assembly nyelvi szint. Az Intel 8086/88 regiszter készlete, társzervezése, címzési módjai, címzési mód byte.
- Az Intel 8086/88 utasításai (adat mozgató, aritmetikai, logikai, string kezelő, bit léptető/forgató, vezérlés átadó, processzor vezérlő, input/output utasítások, program megszakítás, szemafor).
- Pszeudo operátorok. Egyszerű adat definíciós utasítások. Struktúra, rekord (definíció, hívás, mezőre hivatkozás).
- Eljárás (deklaráció, hívás, paraméter átadás/átvétel). Lokális adat terület, rekurzív és reentrent eljárások.
- Feltételes fordítás.
- Makró (definíció, hívás), blokk ismétlés.
- Címkék, változók, konstansok, kifejezések.

Tematika

- Szegmens definíció, szegmens csoport, aktív szegmensek kijelölése, globális szimbólumok.
- Assemblernek szóló utasítások, lista vezérlési operátorok.
- Egyszerűsített lexikális elemző.
- Két menetes assembler fordító.
- Makró generátor.
- Szerkesztő.
- Time sharing (idő osztás). Binding (cím hozzárendelés), dinamikus szerkesztés.
- Programok hangolása.

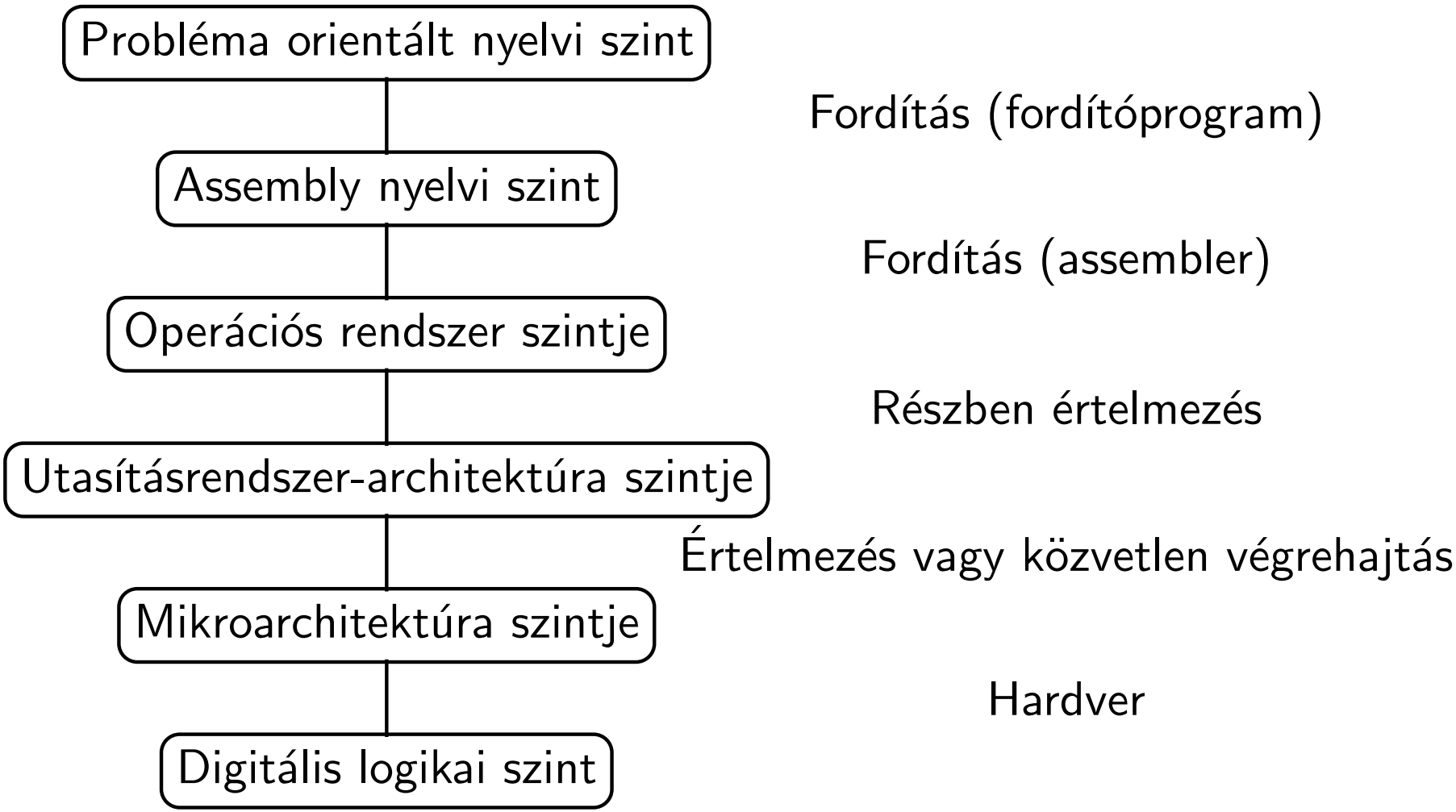
Ajánlott irodalom

- Máté Eörs: Assembly programozás (NOVADAT, 1999, 2000).
- Pethő Ádám: IBM PC/XT felhasználóknak és programozóknak, 1. kötet: Assembly alapismeretek (SZÁMALK, 1992).
- S. Tanenbaum: Structured computer organization (Prentice Hall, 2006). Magyarul: Számítógép-architektúrák 2. átdolgozott, bővített kiadás (Panem 2006).
- B. B. Brey: Programming the 80286, 80386, 80468, and Pentium-baseb Personal Computer (Prentice Hall, 1996).

Követelmények

- Gyakorlat:
  - 1 nagy ZH: 90 perc, 40 pont
  - a gyakorlat sikeres, ha a hallgató legalább 20 pontot szerzett
  - beadható feladat
- Kollokvium:
  - írásbeli vizsga
  - kollokviumjegy: (lásd a CooSpace-ben és az ETR-ben)

Architektúrális szintek



Fordítás és értelmezés

Fordítás: magasabb szintű nyelvek átalakítása numerikus gépi nyelvre vagy annak szimbólikus formájára.
Értelmezés: A utasítás sorozatok beolvasása és végrehajtása.
Assembler: olyan fordító ahol a forrásnyelv szimbólikus formája a gépi nyelvnek.

Magas szintű nyelvek

Változók, függvények, típuskonverziók, polimorgizmus, objektum-orientáltság

```
double sum = 0;
for ( int i = 0; i < 20; i++ ) {
sum += sqrt( (double) i );
}
```

Assembly nyelv

Egysoros utasítások. Az utasítások a gépi kód szimbólikus kódja

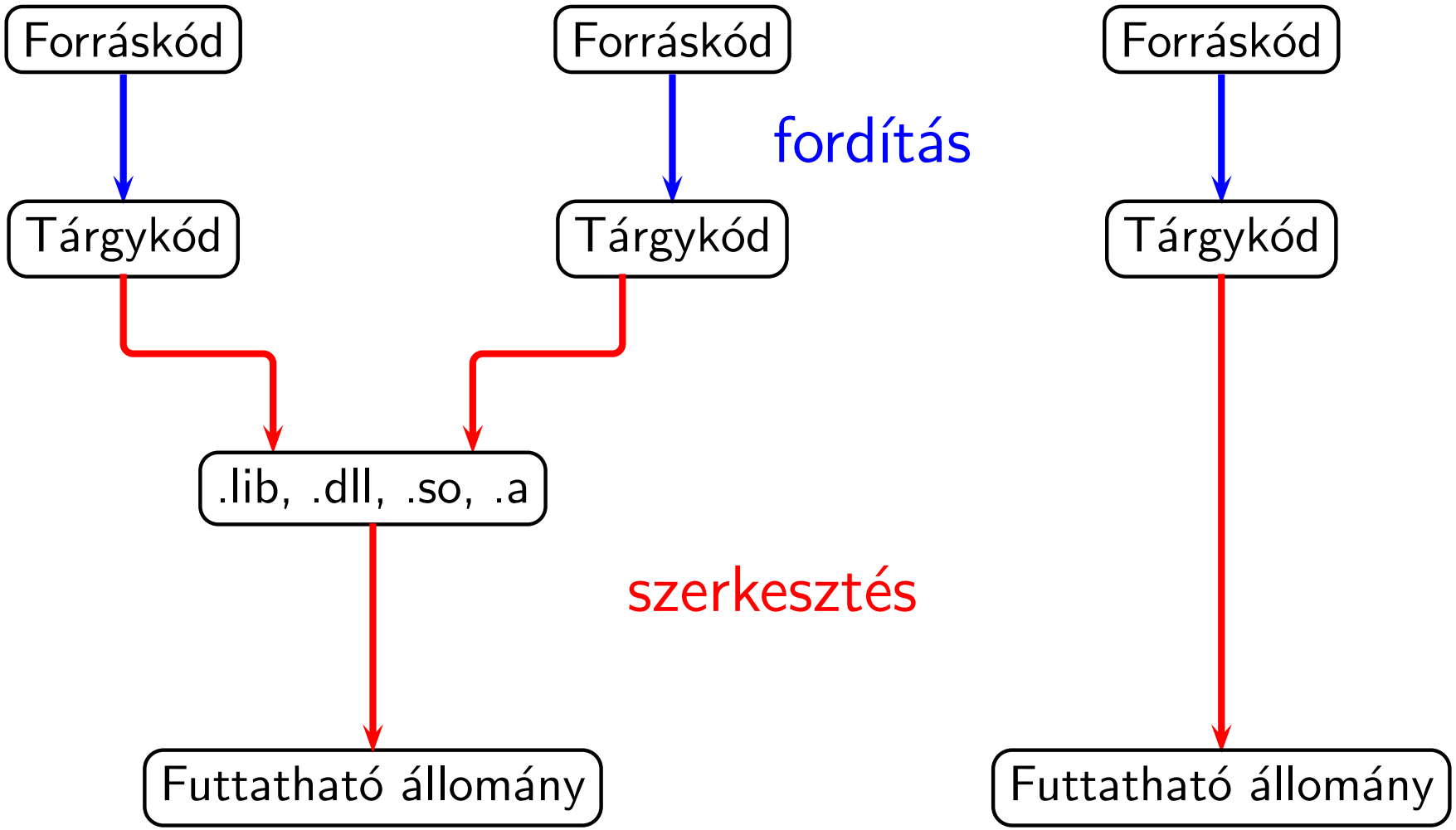
```
MOV AX, 10d ; AX = 10d
ADD AX, [02] ; AX=AX+DS:[02]
PUSH AX ; AX a verembe kerul
```

Gépi nyelv

- A gép számára érthető.
- Gyorsan végrehajtható.
- Processzortípusonként változhat.

```
00 05 0F 2B 42 6F
24 23 94 6E 4A B2
```

Forráskódból futtatható állomány előállítása



Néhány jellemző

- Little endian memória-szervezés
- 16 bites regiszterek
- 20 bites címbusz

Regiszterek

Szegmens regiszterek (16 bites regiszterek)

- CS (Code Segment): utasítások címzéséhez
- SS (Stack Segment): verem címzéséhez
- DS (Data Segment): adat terület címzéshez
- ES (Extra Segment): másodlagos adatterület címzéshez

Regiszterek

Vezérlő regiszterek (16 bites regiszterek)

- IP (Instruction Pointer): az éppen végrehajtandó utasítás logikai címét tartalmazza a CS által mutatott szegmensben
- SP (Stack Pointer): a verem tetejére beírt adat logikai címére mutat az SS által mutatott szegmensben
- STATUS (vagy SR, vagy FLAGS): a processzor állapotát jelző regiszter
- BP (Base Pointer): a verem indexelt címzéséhez használatos
- SI (Source Index): a kiindulási adat terület indexelt címzéséhez használatos
- DI (Destination Index): a cél adat terület indexelt címzéséhez használatos

Regiszterek

Általános regiszterek 16-bites regiszterek.
Az általános regiszterek felső 8 bitje és alsó 8 bitje külön is címezhető

regiszter	felső bájt	alsó bájt	
AX	AH	AL	Accumulator (szorzás, osztás)
BX	BH	BL	Base Register (címző regiszter)
CX	CH	CL	Counter Regiszter (számláló)
DX	DH	DL	Data Register (szorzás, osztás, I/O)

Címzési módok

- Szegmensekre (egybefüggő területekre) osztott memóra.
- A szegmens címek mindig oszhatóak 16-tal (paragrafushatár).
- Virtuális cím: `BÁZISCÍM:OFFSZET`
- Fizikai cím: a fizikai címet a szegmens kezdőcíme és a az offszet együttesen határozza meg.
  `BÁZISCÍM` $\cdot$ `16` + `OFFSZET` alsó 20 bitje. (valós módban)
- Egy fizikai címhez több logikai cím tartozik.

Adat terület címzés

Kódba épített adat Az adatot közvetlenül a regiszterbe írjuk. Ennél a címzési módnál csak konstansokat tudunk megadni második operandusként.

Formátum: `regiszter, konstans`
Például:

- `AX, 6`
- `AX, 06F2h`

Adat terület címzés

Direkt memória címzés A címrészen az operandus logikai (offszet) címét adjuk meg, nem az adatot.
Formátum: `regiszter, memóriacím`
Például:

- `AX, SZO` , *ahol a SZO egy word értéket tartalmaz*
- `AL, KAR` , *ahol a KAR egy byte értéket tartalmaz*

Ügyelni kell arra, hogy a két operandus mérete összhangban legyen egymással. 16 bites regiszterhez csak 16 bites (word) címeket használhatunk, 8 bites regiszterhez csak 8 bites címeket használhatunk!
Direkt címzésnél megadhatunk második operandusként egy regisztert is. Akár egy regisztert, akár egy logikai címet adunk meg, mindkét esetben a memóriacímen tárolt adat változhat, viszont a cím azonos marad.

Adat terület címzés

Indexelt címzés Az operandusban megadott 8 vagy 16 bites számot (eltolás) hozzáadjuk az index-regiszter (SI vagy DI) tartalmához, így alakul ki a logikai cím.
Formátum: `regiszter, szám[index-regiszter]`
Például:

- `AX, [SI]` , *itt csak az SI-ben tárolt értéket számítjuk*
- `AX, 10h[SI]` , *SI tartalmához hozzáadjuk a 10h konstanst*
- `AX, -10h[SI]` , *SI tartalmához hozzáadjuk a -10h konstanst*

A 8 biten megadott eltolás érték előjel helyesen 16 bitre bővül a cím kiszámításakor.

Adat terület címzés

Regiszter indirekt címzés Az operandusban a címet egy regiszter tartalmazza. Az ilyen módon megadott címet mutatónak hívjuk. Nem adható meg eltolás. Ebben a címzési módnál csak az SI, DI és BX regiszter használható.
Formátum: `regiszter, [regiszter]`
Például:

- `AX, [SI]` , *az SI-ben tárolt értéket használjuk*
- `AX, [BX]` , *a BX-ben tárolt értéket használjuk*
- `AX, [DI]` , *a DI-ben tárolt értéket használjuk*

Adat terület címzés

Bázis relatív címzés Az operandusban megadott logikai címet úgy kapjuk meg, hogy az eltolás értékét, az SI vagy DI valamelyikének értékét, valamint a BX regiszterben tárolt értéket összeadjuk.
Formátum: `regiszter, szám[regiszter][regiszter]`
Például:

- `AX, 10h[SI][BX]`
- `AX, [BX][DI]`
- `AX, [BX + SI + 10h]`

Verem terület címzés

A verem (stack) terület címzése bázis relatív címzéssel történik, azzal a különbséggel, hogy BX helyett BP-t használjuk. A fizikai cím meghatározásához nem DS-t, hanem SS lesz a szegmens regiszter.

Program terület címzés

Egy-egy utasítás során az IP értéke az utasítás hosszával növekszik és a soron következő utasításra mutat a CS-ben. Bizonyos vezérlési szerkezetekben (pl. ciklusok, feltételek) az IP értékének megadásával a kívánt helyen folytathatjuk a program futását.

Program terület címzés

IP relatív címzés Az IP (már módosult) pillanatnyi értékéhez hozzáadódik a 8 bites előjeles operandus. A programkódokban bizonyos kódrészletek kezdő utasítását címkével láthatjuk el, ha azt szeretnénk, hogy egy adott feltétel esetén ott folytatódjon a program futása. A feltételes vezérlés átadás és a ciklus utasítás mindig ilyen címzési móddal valósul meg.
Például:
`JMP CIKLUS` , *ahol CIKLUS egy címke az assembly forráskódban*

Program terület címzés

Direkt utasítás címzés Ez a címzési mód közeli (NEAR) vagy távoli (FAR) vezérlés átadásoknál játszik szerepet. A vezérlés átadás közeli, ha a programkód ugyanabban a szegmensben folytatódik tovább és távoli szegmensváltás esetén. Közeli vezérlésátadás esetén a 16 bites operandus lesz az IP új tartalma, távoli vezérlésátadásnál pedig az IP és a CS tartalma is megváltozik. Ilyen vezérlés lehet például az eljáráshívás.
Például:
`CALL ELJARAS` , *az ELJARAS nevű eljárás hívása*

Program terület címzés

Indirekt utasítás címzés Bármilyen címzési móddal megadott szóban vagy dupla szóban tárolt címre történő vezérlés átadás. Például:

- `JMP AX`
- `JMP [BX]`

Az assembly utasítások szerkezete

Assembly programozási nyelvben egy utasítás a következő sémát követi:

```
címrész operációs_kód operandusok kommentár
```

- címrész: egyes adatok illetve utasítások szimbólikus jelölése
- operációs_kód: mnemonic, az utasítás, művelet megnevezésére szolgál
- operandusok: az utasítás paraméterei
- kommentár: A program jobb olvashatóságát és érthetőségét teszi lehetővé, de nincs hatása a program működésére

Példa:

Az assembly utasítások szerkezete

prefixum	operációs kód	címzési mód	operandus
0-2 byte	1 byte	0-1 byte	0-4 byte

- Prefixum: utasítás istmétlés (sztringkezelő utasítások), explicit szegmens megadás (pl. `CS:S`) vagy `LOCK` (szemafor).
- operációs kód: a művelet kódja (a szimbólikus alakja a mnemonic)
- címzési mód byte: hogyan kell az operandust címezni
- operandus: mivel kell az operandust elvégezni

Címzési mód byte

7	6	5	4	3	2	1	0
Mód		Regiszter			Reg/Mem		

Adatmozgató utasítások

- `XCHG`: két operandusa van, amelyek nem lehetnek közvetlen operandusok. Az utasítás kicseréli a két operandusának tartalmát egymással.
- `XLAT`: nincs operandusa. Az `AL`-be a `[BX+AL]` által címzett maximum 256 byte-os tartomány `AL`-edik bájtját tölti.
- MOV: két operandusa van op1 és op2. Az op2 értékét op1-be viszi

Adatmozgató utasítások

- `PUSH`: egyetlen operandusát betölti a veremmutató (`SS:SP`) által mutatott memória címre a verembe, majd a veremmutatót két bájttal csökkenti (a verem „lefelé bővül”).
- `PUSHF`: nincs operandusa, a `STATUS` regiszter értékét tárolja el a verembe (Push Flags).
- `POP`: egyetlen operandusa a veremmutató (`SS:SP`) által címzett értéket másolja az operandusként megadott helyre.
- `POPF`: nincs operandusa, a `STATUS` regiszter értéke lesz a veremmutató által mutatott memóriacím tartalma (Pop Flags). A veremmutató ezután 2-vel növekszik.
- `SAHF`: Az `AH` regiszter tartalma átírodik a `STATUS` regiszter alsó 8 bitjébe.
- `LAHF`: Az `STATUS` regiszter alsó 8 bitje átírodik az `AH` regiszterbe.

A STATUS regiszter

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	O	D	I	T	S	Z	-	A	-	P	-	C

Overflow	Előjeles túlcsordulás
Direction	a string műveletek iránya, 0: növekvő, 1: csökkenő
Interrupt	1: megszakítás engedélyezése, 0: tiltása
Trap	1: „single step”, 0: automatikus üzemmód
Sign	az eredmény legmagasabb helyiértékű bitje (előjel)
Zero	1, ha az eredmény 0, különben 0
Auxiliary Carry	átvitel a 3. és 4. bit között (decimális aritmetika)
Parity	az eredmény alsó 8 bitjének paritása
Carry	átvitel előjel nélküli műveletekhez

Összeadás

- `ADD op1, op2`
- `ADC`: A többszavas összeadást támogatja. Előfordulhat, hogy az összeadás két tagja nem fér el 16 biten, akkor a két operandust a (DX:AX) és a (CX:BX) regiszterpárban tárolhatjuk.
- `AAA`: A pakolatlan (ASCII kódú, egy számjegy nyolc biten) decimális számok összeadását támogatja.
- `DAA`: A pakolt (ASCII kódú, egy számjegy 4 biten) decimális számok összeadását támogatja. Az `A` flag jelzi, ha a művelet során a 3. bitről túlcsordulás történik, ilyenkor `A=1`.

Kivonás

- SUB: két operandusú kivonás művelet, az első operandusból vonja ki a második operandus értékét. Az eredmény az első operandusban keletkezik.
- SBB: A többszavas kivonás műveletét segíti.
- DEC: Az operandusának az értékét 1-gyel csökkenti. Nincs hatással a Carry-re.
- AAS: Két ASCII szám kivonása után korrigál:
  1. Ha AL alsó 4 bitje ≤ 9 és az A flag 0, akkor a 3. lépés jön.
  2. Ha AL = AL-6, AH = AH-1, A = 1
  3. AL felső 4 bitje 0
  4. C = A
- DAS: Két pakolt ASCII szám kivonása után korrigál:
  1. Ha A = 0 vagy az AL alsó 4 bitje ≤ 9, akkor AL = AL-6, A = 1.
  2. Ha C = 1 vagy AL felső 4 bitje ≤ 9, akkor AL = AL - 60h, C = 1.

Szorzás

Assemblyben megkülönböztetünk előjeles és előjel nélküli szorzást.

- `MUL`: Előjel nélküli szorzás. A szorzás műveletét úgy alakították ki, hogy két 8 bites, vagy pedig két 16 bites számot tudjunk szorozni. A `MUL` utasításnak egyetlen operandusa van (op), amely nem lehet közvetlen operandus (vagyis konstans). Két szám szorzatakor egy egyik (első) operandus 8 bites számok esetén az AL-ben tárolódik, 16 bites számok esetén pedig AX-ben.
  - ha op 8 bites, akkor az eredmény AX-ben keletkezik,
  - ha op 16 bites, akkor az eredmény DX:AX-ben keletkezik.

  A `MUL` utasítás hatással van az `O` és `C` flag-ekre.

Szorzás

- A `MUL` utasítás előjel nélküli szorzást valósít meg, amely nincs tekintettel a negatív számokra.

```
MOV AL, -10d        ; AL=0F6h
MOV BL, 02d         ; BL=02h
MUL BL              ; AX=01ECh
                    ; pedig -20=1110 1100 = ECh
```

- `IMUL`: Előjeles szorzás. Működése hasonló a `MUL` utasításhoz.

Osztás

- `DIV`: Előjel nélküli osztás. Egyetlen operandusa van, amely nem lehet közvetlen operandus (vagyis konstans). Ha az op operandus 8 bites, akkor az eredmény úgy áll elő, hogy `AL` tartalmazz az `AX`/op hányadosát, `AH`-ba pedig `AX`/op maradéka kerül. Ha az op operandus 16 bites, akkor `AX`-be a `(DX:AX)`/op hányadosa, `DX`-be pedig a `(DX:AX)`/op maradéka kerül.

Osztás

- `IDIV`: Előjeles osztás. Egyetlen operandusa van, amely nem lehet közvetlen operandus (vagyis konstans). A nem 0 maradék előjele megegyezik az osztóéval. Ha az op operandus 8 bites, akkor az eredmény úgy áll elő, hogy `AL` tartalmazza az `AX/op` hányadosát, `AH`-ba pedig `AX/op` maradéka kerül. Ha az op operandus 16 bites, akkor `AX`-be a `(DX:AX)/op` hányadosa, `DX`-be pedig a `(DX:AX)/op` maradéka kerül.

Osztás

Az osztásnál előfordulhat, hogy a hányados nem fér el `AH`-ban vagy `AX`-ben, ilyenkor azonnal abortál a program. Célszerű megelőző ellenőrzéseket végezni:

- Nem nulla-e az osztó?
- Nem túl nagy-e az osztandó? (Előjel nélküli osztásnál túl nagy az osztandó, ha `AH` $>$ op).

Egyéb műveletek

- `NEG`: Az operandusának ellentettjét számolja ki.
- `CMP`: Összehasonlítás. Két operandusa op1 és op2. op1 értékét op1 - op2 szerint állítja be és a `STATUS` regiszter `Z` flagjét 1-re állítja, ha op1 - op2 = 0
- `AAM`: nincs operandusa. Az `AL` értékét állítja át binárisan kódolt decimális (BCD) értékre `AH`-ba és `AL`-be.

Ciklusok magas szintű programozási nyelvekben

```
int sum = 0;
for ( int i = 0; i < 10; i++ )
{
        sum = sum + i;
}
```

Vezérlés átadás assemblyben

- címkékhez van rendelve a vezérlésátadás
- a `STATUS` regiszter flagjei is döntőek lehetnek
- eljárásokkal kapcsolatos vezérlésátadás.

Feltétlen vezérlésátadás (ugrás)

`JMP op`

- ha op közeli, akkor `IP = op`
- ha op távoli, akkor `(CS:IP) = op`

KERÜLENDŐ!!!

Mi lesz az összehasonlítás eredménye?

```
start:  MOV AX, 50d
        ADD AX, 02d
        JMP ide
        ADD AX, 22d
ide:    CMP AX, 74d

        ; Z = ?
```

Feltételes vezérlés átadás (ugrás)

- aritmetikai csoport: `CMP`, `SUB`
  megkülönböztetjük az előjeles és az előjel nélküli utasításokat
- logikai csoport: a flag-ek állapota dönt

Feltételes ugrások, aritmetikai csoport

Előjeles			Reláció	Előjel nélküli				
JZ	$\equiv$	JE	$=$	JZ	$\equiv$	JE		
JNZ	$\equiv$	JNE	$\neq$	JNZ	$\equiv$	JE		
JG	$\equiv$	JNLE	$>$	JA	$\equiv$	JNBE		
JGE	$\equiv$	JNL	$\geq$	JAE	$\equiv$	JNB	$\equiv$	JNC
JL	$\equiv$	JNGE	$<$	JB	$\equiv$	JNAE	$\equiv$	JC
JLE	$\equiv$	JNG	$\leq$	JBE	$\equiv$	JNA		

Feltételes ugrások, logikai csoport

a flag igaz (1)			flag	a flag hamis (0)		
JZ	≡	JE	Zero	JNZ	≡	JNE
JC			Carry	JNC		
JS			Sign	JNS		
JO			Overflow	JNO		
JP	≡	JPE	Parity	JNP	≡	JPO
JCXZ			CX = 0			

Példa feltételes ugrásra

Példaprogramok az előjel meghatározására.

```
int szam = -5;
int sign = 0;
if (szam < 0) {
        sign = -1;
} else if (szam > 0) {
        sign = 1;
}
```

```
        MOV CX, 0
        CMP AX, 0
        JE vege
        JG poz
        JL neg
poz:    MOV CX, 1d
        JMP vege
neg:    MOV CX,-1d
vege:   ...
```

Ciklus szervező utasítások

- relatív címzéssel valósulnak meg,
- csak `SHORT` ugrásra képesek

`LOOP`	`ipr`	CX-et eggyel csökkent, majd ugrás az `ipr`-re, ha CX $\neq 0$
`LOOPZ`	`ipr`	CX-et eggyel csökenti, majd ugrás az `ipr`-re, ha CX $\neq 0$ és `Z=1`
`LOOPE`	`ipr`	ugyanaz, mint a `LOOPZ`
`LOOPNZ`	`ipr`	CX-et eggyel csökkenti, majd ugrás az `ipr`-re, ha CX $\neq 0$ és `Z=0`
`LOOPNE`	`ipr`	ugyanaz, mint a `LOOPNZ`

Ciklus szervező utasítások, példa

Példaprogramok az első 10 pozitív egész szám összegének meghatározására.

```
int n = 10;
int sum = 0;
while (n > 0) {
      sum = sum+n;
}
```

```
        MOV CX, 10d
        MOV AX, 0
ciklus: ADD AX, CX
        LOOP ciklus
vege:   ...
```

String kezelő utasítások

- byte-okból vagy word-ökből álló összefüggő adat terület címzése
- a forrás cím mindig (`DS:SI`)
- a cél mindig (`ES:DI`)
- nem kötelező a műveletek operandusait megadni (a mnemonic utal arra, hogy byte-os vagy word-ös műveletet kell végeznünk)
- minden utasítás módosítja vagy az `SI` vagy a `DI` indexregiszter értékét a `D` (Direction) flagnek megfelelően: `D=0` növekvő, `D=1` csökkenő.

String kezelő utasítások

Byte illetve word átvitele a (`DS:SI`) által mutatott címről a (`ES:DI`) által mutatott címre.

`MOVSB`		MOVe String Byte
`MOVSW`		MOVe String Word
`MOVS`	`d,s`	MOVe String (byte vagy word)

String kezelő utasítások

Betöltés `AL`-be illetve `AX`-be a (`DS:SI`) által mutatott címről:

`LODSB`		LOaD String Byte
`LODSW`		LOaD String Word
`LODS`	`s`	LOaD String (byte vagy word)

String kezelő utasítások

Tárolás az (`ES:DI`) által mutatott címre `AL`-ből, illetve `AX`-ből:

`STOSB`		STOre String Byte
`STOSW`		STOre String Word
`STOS`	`d`	STOre String (byte vagy word)

String kezelő utasítások

Az (`ES:DI`) és a (`DS:SI`) által mutatott címen lévő byte illetve szó összehasonlítása:

`CMPSB`		COMPare String Byte
`CMPSW`		COMPare String Word
`CMPS`	`d,s`	COMPare String (byte vagy word)

A flag-ek `s - d` értékének megfelően állítódnak be.

String kezelő utasítások

Az (`ES:DI`) által mutatott címen lévő byte illetve szó összehasonlítása `AL`-lel illetve `AX`-szel. A flag-ek `AX - d` értékének megfelelően állnak be.

`SCASB`		SCAn String Byte
`SCASW`		SCAn String Word
`SCAS`	`d`	SCAn String (byte vagy word)

String kezelő utasítások

A string kezelő utasítások elé ismétlő prefixumokat írhatunk, és ilyenkor ezek az utasítások többször kerülhetnek végrehajtásra.

- `REP` ≡ `REPZ` ≡ `REPE`
- `REPNZ` ≡ `REPNE`

Működése hasonló a `LOOP` utasításhoz.

- ha `CX = 0`, akkor a string kezelő utasítás nem hajtódik végre
- különben minden utasítás végrehajtásánál `CX` tartalma eggyel csökken
- ha a string kezelő utasítás beállítja a flag-eket, akkor az ismétlés feltétele, hogy a flag állapota megegyezzen a prefixum végződésében előírttal.

Példa

Van-e az array tömbnek 3-tól különböző eleme?

```
          MOV CX, 100
          MOV DI, -1
          MOV AL, 3
ciklus:   INC DI
          CMP array[DI], AL
          LOOPE ciklus
          JNE NEM3
          ...
nem3:     ...
```

```
          MOV CX, 100
          MOV DI, offset
          array
          MOV AL, 3
REPE      SCAS array
          JNE NEM3
          ...
nem3:     DEC DI
```

Gyorsabb

Logikai műveletek

`XOR`	`op1, op2`	KIZÁRÓ VAGY művelet
`AND`	`op1, op2`	bitenkénti ÉS művelet
`OR`	`op1, op2`	bitenkénti VAGY művelet
`TEST`	`op1, op2`	a flag-ek `op1 & op2` szerint
`NOT`	`op1`	bitenkénti negáció

- a `NOT` utasítás nem módosítja a `STATUS` tartalmát
- a logikai utasítások nem módosítják az `O` és `C` flag-ek értékét

Bitforgató és léptető utasítások

A forgatás/léptetés történhet 1 bittel vagy vagy byte illetve word esetén a CL regiszter alsó 3 illetve 4 bitjén meagott bittel jobbra vagy balra.

`RCR`	`op,1/CL`	forgatás jobbra a Carry-n keresztül
`RCL`	`op,1/CL`	forgatás balra a Carry-n keresztül
`ROR`	`op,1/CL`	forgatás jobbra
`ROL`	`op,1/CL`	forgatás balra
`SHR`	`op,1/CL`	léptetés jobbra
`SHL`	`op,1/CL`	léptetés balra
`SAR`	`op,1/CL`	aritmetikai léptetés jobbra (op előjele lesz a belépő bit)
`SAL`	`op,1/CL`	aritmetikai léptetés balra (op előjele lesz a belépő bit)

`SAL` = `SHL`

Processzor vezérlő utasítások

A `STATUS` regiszter bitjeit módosíthatják:

`STC`	Set `C`
`CLC`	Clear `C`
`STD`	Set `D`
`CLD`	Clear `D`
`STI`	Set `I`
`CLI`	Clear `I`
`CMC`	Complement `C`

Processzor vezérlő utasítások

`NOP`	üres utasítás, nem végez műveletet
`WAIT`	a processzor külső megszakításig várakozik
`HLT`	leállítja a processzort

Megszakítások

- használható BIOS rutinok meghívásra,
- debug módban (`INT 3`, az egyetlen egybájtos megszakításkód),
- kilépéshez

`INT`	`i`	megszakítás az `i`-edik ok szerint
`INTO`		megszakítás, ha `O=1`
`IRET`		visszatérés megszakító rutinból, `IP`, `CS`, `STATUS` feltöltése a veremből

Eljárások

- közeli (NEAR) és távoli (FAR) eljárások
- a főprogram eljárás mindig távoli
- a közeli eljárások ugyanazon szegmensen belül kezdődnek, a távoli eljárások másik szegmensben kezdődnek

Eljárás deklaráció

```
eljaras_nev   PROC   típus     ; típus:  NEAR/FAR
              ...
              ...
              RET               ; visszateres
eljaras_nev   ENDP
```

Eljárás hívás és visszatérés

- `CALL` *eljaras_nev* : közeli eljáráshíváskor az `IP`, távoli eljáráshíváskor az `IP` és `CS` tartalma a verembe kerül
- `RET` : a `RET` utasítás hatására a `IP` (vagy ha az eljárás távoli volt, akkor `IP` és `CS` tartalma kikerül a veremből, és a program futása az eljáráshívás utáni utasítással folytatódik

Paraméterek átadása

- szabványos helyen történő átadás: Építsük be a forráskódba. Hozzunk létre egy címet, ahol az adat megtalálható és ovassuk onnan az adatot
- regiszterekben történő átadás: a paraméterek címét vagy értékét a megfelelő regiszterekbe tároljuk el az eljáráshívás előtt, majd használjuk azt
- paraméterek veremben történő átadása: a paramétereket meghatározott sorrendben a verembe tároljuk el

Paraméterek szabványos területen történő átadása – 1

```
skalar   PROC    FAR                ; foprogram
         ...                        ; elokeszites
         CALL    skal               ; meghivjuk az eljarast
         ...
skalar   ENDP
```

Paraméterek szabványos területen történő átadása – 2

```
skal    PROC                    ; eljaras
        PUSH    BX              ; mentesek
        PUSH    CX
        PUSH    DX
        MOV     CL,n            ; n az adatszegmensben van, byte
        XOR     CH, CH          ; CX = n
        XOR     DX, DX          ; DX = 0
        JCXZ    kesz            ; ugras a kesz cimkere, ha CX = 0
        XOR     BX, BX          ; BX = 0, index lesz
ism:    MOV     AL, a[BX]       ; a egy vektor címe az adatszegmensben
        IMUL    b[BX]           ; b egy vektor címe az adatszegmensben
        ADD     DX, AX          ; reszosszeg
        INC     BX
        LOOP    ism
kesz:   MOV     AX, DX          ; a skalar erteke AX-ben
        POP     DX              ; visszamentesek
        POP     CX
        POP     BX
        RET                     ; visszateres
skal    ENDP
```

Paraméterek regiszterekben történő átadása – 1

```
skalar  PROC    FAR             ; foprogram
        ...                     ; elokeszites
        MOV     CL, n
        XOR     CH, CH          ; CX = n
        MOV     SI, OFFSET a    ; SI-be az a vektor cime
                                kerul
        MOV     DI, OFFSET b    ;DI-be a b vektor cime kerul
        CALL    skal            ; meghivjuk az eljarast
        ...
skalar  ENDP
```

Paraméterek regiszterekben történő átadása – 2

```
skal    PROC                        ; eljaras
        PUSH    BX                  ; mentesek
        PUSH    CX
        PUSH    DX
        XOR     DX, DX              ; DX = 0
        JCXZ    kesz                ; ugras a kesz cimkere,
                                    ; ha CX = 0
        XOR     BX, BX              ; BX = 0, index lesz
ism:    MOV     AL, [SI+BX]         ; a[i]
        IMUL    BYTE PTR [DI+BX]    ; b[i]
        ADD     DX, AX              ; reszosszeg
        INC     BX
        LOOP    ism
kesz:   MOV     AX, DX              ; a skalar erteke AX-ben
        POP     DX                  ; visszamentesek
        POP     CX
        POP     BX
        RET                         ; visszateres
skal    ENDP
```

Paraméterek veremben történő átadása – 1

```
skalar  PROC    FAR                 ; foprogram
        ...                         ; elokeszites
        MOV     AL, n
        XOR     AH, AH              ; AX = n
        PUSH    AX
        MOV     AX, OFFSET a
        PUSH    AX
        MOV     AX, OFFSET b
        CALL    skal                ; meghivjuk az eljarast
        ADD     SP, 6
        ...
skalar  ENDP
```

Paraméterek regiszterekben történő átadása – 2

```
skal    PROC                        ; eljaras
        PUSH    BP                  ; mentesek
        MOV     SP, BP
        PUSH    SI
        PUSH    DI
        PUSH    BX
        PUSH    CX
        PUSH    DX
        MOV     SI, 6[BP]           stack rel. cimzes
        MOV     DI, 4[BP]           stack rel. cimzes
        MOV     CX, 8[BP]           stack rel. cimzes
        XOR     DX, DX              ; DX = 0
        XOR     BX, BX              ; BX = 0, index lesz
        JCXZ    kesz                ; ugras a kesz cimkere,
                                    ; ha CX = 0
ism:    MOV     AL, [SI+BX]         ; a[i]
        IMUL    BYTE PTR [DI+BX]    ; b[i]
        ADD     DX, AX              ; reszosszeg
        INC     BX
        LOOP    ism
```

Paraméterek regiszterekben történő átadása – 3

```
kesz:   MOV     AX, DX                  ; a skalar erteke AX-ben
        POP     DX                      ; visszamentesek
        POP     CX
        POP     BX
        POP     DI
        POP     SI
        POP     BP
        RET                             ; visszateres
skal    ENDP
```

Rekurzív és reentrent eljárások

- Rekurzív eljárás: önmagát hívó eljárás
- Reentrent eljárás: többszöri belépést tesz lehetővé, akár véletlenszerűen.

A lokális adat a veremben alakítható ki:

```
eljaras   PROC    ...
          PUSH    BP
          MOV     BP, SP
          SUB     SP, n     ; n word meretu teruletet foglalunk
          ...               ; tovabbi regiszter mentesek
          ...
          MOV     SP, BP    ; visszamentes
          RET     ...       ; visszamentes
```

Egyszerű adatdefiníciós utasítások

- `n db 25 ; 1 byte, kezdőértéke 25 decimális`
- `a db 4h ; 1 byte, kezdőértéke 4 hexadecimális`
- `p db 1,2,3 ; 3 byte`
- `kar db 'a','b','c' ; 3 ASCII karakter`
- `szoveg db "Hello world!", 13, 0AH`
  `; ASCII kódú szöveg és két szám`
- `szoveg2 db "Hello","world!"`
- `szo dw 053AH ; 1 szó`
- `szo_cime dw szo ; a szo címe`

Struktúra

```
points  STRUCT
x       db 5 ; felulirhato
y       db 6 ; felulirhato
rgb     db 213,25,150 ; nem felulirhato
label   db 'b','e' ; nem felulrihato
name    db 'point'; felulirhato
points  ENDS
```

Struktúra

- Struktúra hívás:

```
P1      POINTS ; kezdoertek a definiciobol
P2      POINTS <0,0,0, , , 'origo'>
P3      POINTS <1,1,1>
P_vec   POINTS 10 dup (<0,0,0>)
```

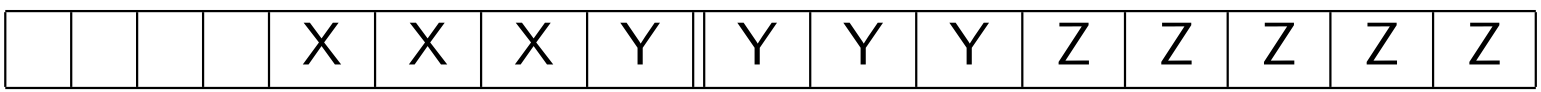
- Struktúra mezőre hivatkozás:
  - `MOV AL, POINTS.x`
  - `MOV BX, OFFSET P1`
  - `MOV AL, [BX].x`
  - `MOV AL, x.[BX]`
  - `MOV AL, [BX] + x`
  - `MOV AL, x [BX]`

Rekord

A rekord egy 1 vagy 2 bájtos összetett adat, amelynek meghatározott bit sorozatai logikai egységet alkotnak. Ezeket mezőknek nevezzük.
Hasznos, ha 1 vagy 2 bájtos adatot bitcsoportként kezelünk (pl. drivereknél).

Rekord

- `rec_tipus RECORD` mező specifikációk
- mező specifikáció:
  `mezo_nev: szelesseg=kezdoertek`
- `R RECORD X:3,Y:4=15,Z:5`

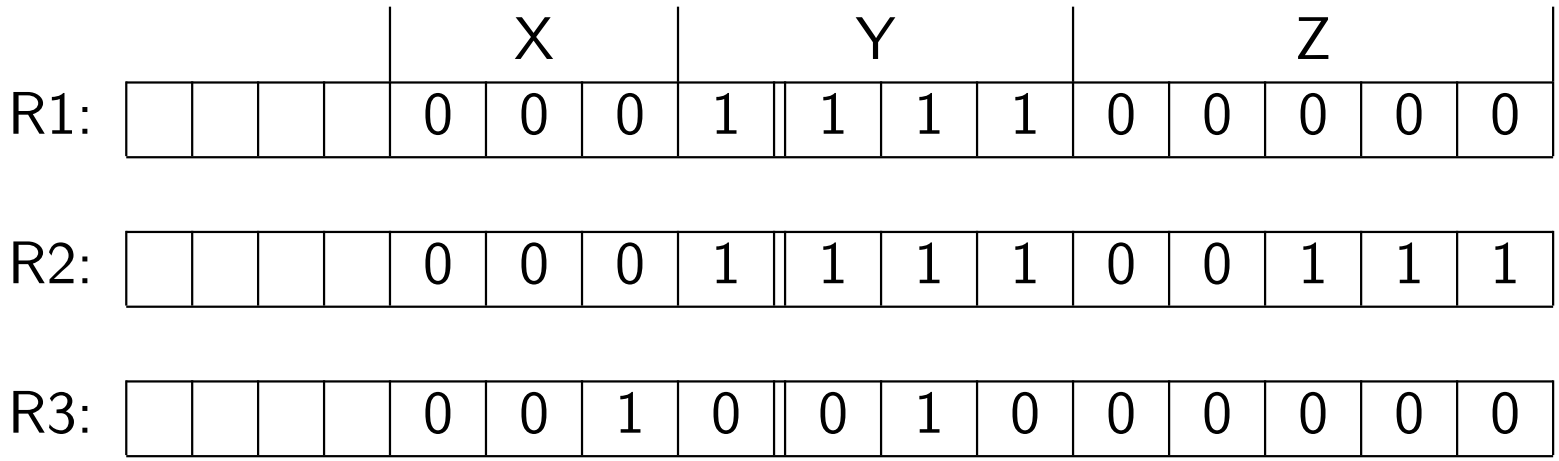


Rekord

- Rekord példányok:

```
R1      R < > ; kezdoertekek a definiciobol
R2      R <,,7> ; Z erteke 7
R3      R <1,2> ; X kezdoerteke 1, Y = 2
R_vec   R 10 dup (<1,2,3>)
```

- Rekord mezők:



Rekord

Rekord mezőre hivatkozás:
Az R3 rekord Y mezejét szeretnénk AX-be írni

```
MOV   AX, R3        ; szavas rekord
AND   AL, MASK Y    ; az Y mezo bitjeinek maszkolasa
MOV   CL, Y         ; leptetes elokeszitese
SHR   AX, CL        ; leptetes, kesz vagyunk
```

Kifejezések

- Konstansok:
  `.RADIX n`
- Szimbólumok:
  - Szimbólikus konstansok
    `S = 1 ; S erteke 1` (megváltoztatható)
    `N EQU 14 ; N erteke 14` (nem változtatható meg)
    ...
    `S = 6; helyes ertekadas`
    `S EQU 6 ; helytelen`
    `N = 3 ; helytelen`
    `N EQU 8 ; helytelen`
  - Címkék

Kifejezés

- szimbólumokból és konstansokból épülnek fel
- az operátrok, pszeudo operátorok operandus része írható
- `[BX]`
- Egy kifejezés akkor megengedett, ha az értéke fordítási időben meghatározható és szintaktikailag helyes

Műveletek 1.

- `()` vagy `[]` : zárójelezés vagy [] jelenthet indirekciót is, ha megengedett
- `.` (pont) : struktúra mező hivatkozás
- `LENGTH változó`: a `változó`-hoz tartozó adatterület elemeinek száma
- `SIZE változó`: a `változóhoz tartozó adatterület hossza bájtban`
- `WIDTH R/F`: az R rekord vagy az F (rekord) mező szélessége bitekben
- `MASK F`: az F (rekord) mező bitjein 1, a többin 0

Műveletek 2.

- Explicit szegmens megadás
- pl.: `MOV AX, ES:[BX] ; AX = az (ES:BX)-en levo szo-val`

Műveletek 3.

- `tipus PTR cim`: típus definiálás (BYTE, WORD, DWORD, QWORD, TBYTE, NEAR, FAR)
  pl.: `MUL BYTE PTR [BX]`
- `OFFSET szimbolum`: a `szimbolum` OFFSET címe
- `SEG szimbolum`: a `szimbolum` szegmens címe
- `TYPE valtozo`: a változó elemeinek hossza bájtokban
- `...  THIS tipus`: a program szöveg adott pontján adott típusú szimbólum létrehozása
  pl.: ADATB EQU THIS BYTE

Műveletek 4.

- `LOW valtozo`: egy szó `valtozo` alsó bájtja
- `HIGH valtozo`: egy szó `valtozo` felső bájtja

Műveletek 5.

Multiplikatív műveletek:

- *: szorzás
- /: osztás
- `MOD`: maradékos osztásnál a nem negatív maradék
- `kifejezes SHL lepes`: a `kifejezes` léptetése balra `lepes` bittel
- `kifejezes SHR lepes`: a `kifejezes` léptetése jobbra `lepes` bittel

Műveletek 6.

Additív műveletek:

- +: összeadás
- -: kivonás

Műveletek 7.

Relációs operátorok:

- EQ: $=$
- NE: $\neq$
- LT: $<$
- LE: $\leq$
- GT: $>$
- GE: $\geq$

Műveletek 8, 9, 10 11.

- `NOT`: bitenkénti negálás
- `AND`: bitenkénti ÉS
- `OR`, `XOR`: bitenkénti VAGY illetve KIZÁRÓ VAGY
- `SHORT`: 8 bites IP relatív címzés kikényszerítése

Feltételes fordítás

```
IFxx    feltetel
        ...             ; lefordul, ha a feltetel igaz
ELSE                    ; elmaradhat
        ...             ; lefordul, ha feltetel hamis
ENDIF
```

Feltételes fordítás

`IFxx`:

- `IF1` : igaz a fordítás első menetében
- `IF2` : igaz a fordítás második menetében
- `IFDEF szimbolum` : igaz, ha a szimbólum definiált
- `IFNDEF szimbolum`: igaz, ha szimbólum nem definiált
- `IFB <arg>`: igaz, ha ¡arg¿ üres (blank)
- `IFNB <arg>`: igaz, ha ¡arg¿ nem üres
- `IFIDN <arg1>,<arg2>` : igaz, ha arg1=arg2
- `IFDIF <arg1>,<arg2>` : igaz, ha arg1≠arg2

Makrók

- gyakran végrehajtandó programrészletek
- a makró definíció nem része a programnak (helyettesítés)
- hatékonyabb az eljárásoknál

Makrók

- Makró definíció:

```
M_nev   MACRO [fpar1][,fpar2] ...   ; makro fej kezdet
        ...                         ; makro torzs
        ENDM                        makro vege
```

- Makró hívás:

```
M_nev [apar1] [,apar2] ...
```

Dupla szavas összeadás: Makró vs. Eljárás

Eljárás

```
EDADD   PROC NEAR
        ADD AX,BX
        ADC DX,CX
        RET
EDADD   ENDP
```

```
CALL EDADD
```

Makró

```
MDADD   MACRO
        ADD AX, BX
        ADC DX, CX
        ENDM
```

```
MDADD
```

Cimke makrón belül

```
KOPOG   MACRO n
        LOCAL ujra
        MOV CX, n
ujra:   KOPP
        LOOP ujra
        ENDM
```

Blokk ismétlés

```
REPT n
...         ; ez imetlodik
ENDM
```

Blokk ismétlés argumentumlista szerint

```
IRP   par, <arg1 [,arg2 ...]>
...   ; ez a resz kerul tobbszor
      ; bemasolasra ugy, hogy
      ; par rendre felveszi
      ; az argumentumokat
ENDM
```

Blokk ismétlés string alapjan

```
IRPC  par,string
...   ; ez a resz kerul tobbszor
      ; bemasolasra ugy, hogy
      ; par rendre felveszi
      ; a string karaktereit
ENDM
```

Kétmenetes fordítás

1. Összegyűjti és táblázatba foglalja a szimbólum definiciókat.
2. Fordítási kísérlet egy menetben. Késleltetni kell a fordítást ott, ahol előre hivatkozás van, pl. táblázatba tenni a még le nem fordított részeket. A második menet végén már minden szimbólum ismert, ekkor fel kell dolgozni a táblázatot.

A kétmenetes fordítás eredménye a tárgy kód (object code).

Szerkesztő

- a programhoz tartozó object file-okat egyetlen futtatható állománnyá szerkeszti
  - az azonos nevű és osztályú szegmens szeleteket összeilleszti a szegmens szeletek definíciójában megadott módon
  - a `GROUP` pszeudo utasítással egy csoportba sorolt szegmensek egymás után helyezése
  - relokáció elvégzése
  - külső hivatkozások (`EXTRN`, különböző modulokban definiált) feloldása
- két menet: táblázat készítés + szerkesztés