HERNECZKI ISTVÁN

MACRO ASSEMBLER

LAPOZGATÓ
SOROZAT

Josef Havel
Max - Stromeyer - Str. 9
7750 Konstanz

Herneczki István

MACRO Assembler

Herneczki István

MACRO Assembler

Műszaki Könyvkiadó ■ Novotrade Rt.

Budapest, *1988.*

LAPOZGATÓ SOROZAT

Lektorálta:

PETHŐ ÁDÁM
okl. matematikus

BORS GYÖRGY
okl. matematikus

Szerkesztő:

GYÖRKE TIBORNÉ
okl. villamosmérnök

ETO: 681.3.06
ISBN: 963 10 7739 X

Tartalom

Az assembler nyelv elemei

AZ ASSEMBLER PROGRAMSOR FELÉPÍTÉSE

[név] műveleti kód operandus(ok) [;megjegyzés]

Hossza maximum 132 karakter.
A sor elemeit legalább 1 szóközzel vagy tabulátorral kell elválasztani.

Név	egy szimbólum. A szimbólum a program elemeit (adatelemek, szegmensek, procedúrák, címkék stb.) azonosítja. Állhat: – betűkből (A–Z), – számjegyekből (0–9), – speciális karakterekből (? . @ _ $). A szimbólum első karaktere nem lehet számjegy, és csak az első 31 karakter értelmezett.
Műveleti kód	mnemonikus kód vagy direktíva.
Operandus	szimbólum, konstans érték vagy ezekből operátorokkal képzett kifejezés.
Megjegyzés	tetszőleges szöveg, amely a ;jeltől a sor végéig tart.

Konstans

Számérték, amelynek nincsen más attribútuma csak az értéke.

Fajtái:

bináris	10110110B,
decimális	256 vagy 256D,
hexadecimális	55H, 0FEH, 0ABCDH,
oktális	3457Q vagy 3457O,
ASCII	'ABC' vagy "ABC",
lebegőpontos (decimális)	34.25E–3,
hexadecimális valós	8-, 16- vagy 20-jegyű hexadecimális szám a vezető 0 nélkül számolva. Például 0FFFFFFFFR.

Területfoglaló direktívával kijelölt memóriaterület, amelylyel az utasítások műveletet végeznek. Az utasítások memóriaoperandusai lehetnek.

Attribútumai:

szegmens 16 bites, csak futási időben ismert érték (szegmenskezdőcím/16), amely megadja, hogy az adatelemet (változót) melyik szegmensben definiáltuk.

offset 16 bites érték. Az adatelem (változó) távolsága a szegmens elejétől byte-ban mérve.

típus konstans érték, amelyet a változó definiálásakor használt területfoglaló direktíva határoz meg.

BYTE 1,
WORD (szó) 2,
DWORD (duplaszó) 4,
QWORD (négy szó) 8,
TBYTE (tíz byte) 10,
STRUC (struktúra) a programozó szabja meg,
RECORD (rekord) 1 vagy 2.

Programterületen levő utasítást azonosít.
A JMP, a CALL és a feltételes ugrási utasítások operandusa.

Attribútumai:

szegmens	ugyanaz, mint a változó esetén.
offset	ugyanaz, mint a változó esetén.
típus	megszabja, hogyan hivatkozhatunk a címkére. **NEAR:** közeli, szegmensen belüli vezérlésátadással, **FAR :** távoli, szegmensközi vezérlésátadással.

Megadható:
utasítássor elején kettősponttal (NEAR típusú),

Példa **CIMKE1: MOV AX,MEM_SZO**

LABEL direktívával,

PROC direktívával.

Szimbólumok és konstans értékek különböző operátorokkal összekapcsolva.
Az assembler a kifejezéseket balról jobbra haladva értékeli ki. A magasabb precendenciájú (rangú) operátorokat előbb veszi figyelembe.

Operátorok (csökkenő precedenciaszintek szerint)

1. Zárójel	(...) vagy [...].
. (pont)	struktúranév.elemnév.
LENGTH változónév	egy változó számára lefoglalt terület elemeinek számát adja vissza.
SIZE változónév	egy változó számára lefoglalt terület byte-ban kifejezett méretét adja vissza (SIZE = LENGTH * TYPE).
Példa	**TER DW 100 DUP(?)** **LENGTH TER = 100** **TYPE TER = 2** **SIZE TER = 200**
WIDTH mezőnév	egy rekordmező bitekben kifejezett szélességét adja vissza (l. RECORD direktíva).
MASK mezőnév	olyan bitmaszkot ad vissza, amely a megadott mező bitjei helyén 1-eket, egyébként 0-kat tartalmaz (l. RECORD direktíva).

2. szegmensregiszter:címkifejezés

vagy

szegmensnév:címkifejezés

vagy

csoportnév:címkifejezés

megadja, hogy melyik szegmensregisztert kell használni a cím kiszámításához a megadott változónál.

Példa **MOV AX,ES:TOMB**

3. típus **PTR** kifejezés átdefiniálja egy változó vagy egy címke típusát. A típus lehet:
BYTE,
WORD,
DWORD,
QWORD,
TBYTE,
ill. NEAR,
FAR.

Példák
```
ADAT DB 1,2
MOV BX,WORD PTR ADAT
INC BYTE PTR [SI]
JMP DWORD PTR [BX]
```

OFFSET változónév vagy címke

egy változó vagy egy címke offsetjét adja vissza.

SEG változónév vagy címke

egy változó vagy egy címke szegmensértékét adja vissza.

TYPE változónév vagy címke

egy változó számára lefoglalt terület elemeinek hosszát adja vissza (vö. LENGTH, SIZE).

THIS típus — adott offsetű és szegmensű helyen a megadott típusú operandust hozza létre. A típus lehet:
BYTE,
WORD,
DWORD,
QWORD,
TBYTE,
ill. **NEAR,**
FAR.

Példa

```
BYTE_TOMB THIS BYTE
SZO_TOMB DW 100 DUP(?)
```

4. **HIGH** és **LOW** — egy szó felső, ill. alsó byte-jára címez.

Példa

```
ADAT DW ?
MOV AH,HIGH ADAT
```

5. Szorzás, osztás, modulo és eltolás (*,/,MOD,SHL,SHR) operátorok.

Példák

```
MOV AX,100 MOD 17    ;AX-be 15
MOV AX,101B SHL (2*2);AX-be 01010000B
```

6. Összeadás, kivonás és negatív előjel (+,–).

7. **EQ** (egyenlő), **NE** (nemegyenlő), **LT** (kisebb, mint), **LE** (kisebb egyenlő), **GT** (nagyobb, mint), **GE** (nagyobb egyenlő) relációs operátorok.

8. **NOT** (logikai NEM).

Példa

```
AND AX,NOT 000FH
```

9. **AND** (logikai ÉS).

Példa

```
MOV AX,0234H AND 00FFH
```

10. **OR** (logikai VAGY), **XOR** (logikai kizáró VAGY).

11. **SHORT** rövid ugrást jelöl ki Ugrási tartomány: előre + 127 byte, vissza 128 byte.

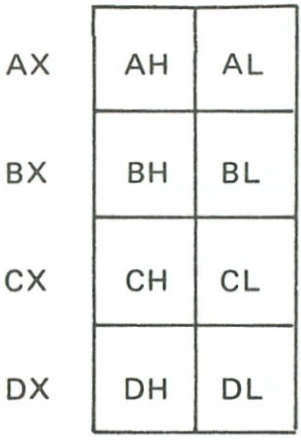
Példa

```
JMP SHORT CIMKE
```

Az Intel 8088 processzor általános tudnivalói

AZ INTEL 8088 REGISZTEREI

Általános regiszterek



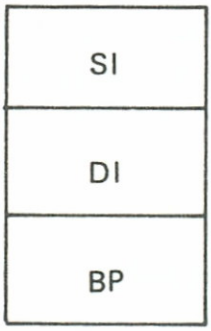
Akkumulátorregiszter
(*A*ccumulator Register).

Bázisregiszter
(*B*ase Register).

Számlálóregiszter
(*C*ount Register).

Adatregiszter
(*D*ata Register).

Címzési regiszterek



Indexregiszter
(*S*ource *I*ndex Register).

Indexregiszter
(*D*estination *I*ndex Register).

Stack- (verem-) bázisregiszter
(*B*ase *P*ointer Register).

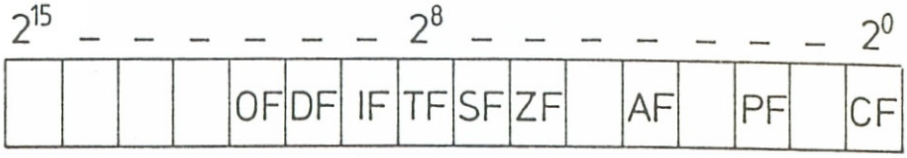
Vezérlőregiszterek

SP	Stackmutató (*S*tack *P*ointer).
IP	Utasításmutató (*I*nstruction *P*ointer).
Flagszó	L. külön részletezve.

Szegmensregiszterek

CS	Kódszegmens-regiszter (*C*ode *S*egment Register).
DS	Adatszegmens-regiszter (*D*ata *S*egment Register).
ES	Második adatszegmens-regiszter (*E*xtra *S*egment Register).
SS	Stackszegmens-regiszter (*S*tack *S*egment Register).

A FLAGSZÓ FELÉPÍTÉSE



CF (*C*arry *F*lag) átvitel. Ha az eredmény legmagasabb helyi értékű bitjénél átvitel keletkezik, CF=1.

PF (*P*arity *F*lag) PF=1, ha az eredmény alsó byte-jában az egyesek száma páros.

AF (*A*uxiliary Carry *F*lag) segédátvitel. Átvitel a 3. bitről a 4. bitre. A decimális korrekciós műveletek használják.

ZF (*Z*ero *F*lag) Ha az eredmény 0, ZF=1.

SF (*S*ign *F*lag) előjel. Az eredmény legmagasabb helyi értékű bitjének tartalma. Előjeles számokkal végzett műveleteknél SF=0 pozitív, ill. 0, SF=1 negatív eredményt jelez.

TF (*T*rap *F*lag) TF=1 lépésenkénti utasításvégrehajtást eredményez (INT 1).

IF (*I*nterrupt *F*lag) IF=1 engedélyezi, IF=0 tiltja a külső maszkolható megszakítások érvényre jutását.

DF (*D*irection *F*lag) műveletvégzési irány. String (füzér) műveleteknél a pointerek (mutatók) automatikusan nőnek, ha DF=0, egyébként csökkennek.

OF (*O*verflow *F*lag)

túlcsordulás. OF=1, ha az előjeles számokkal végzett műveletek eredménye nem fér el az ábrázolási területen (indokolatlan előjelváltás). Értéke az aritmetikai műveletek során a legmagasabb helyi értékről kilépő, ill. a legmagasabb helyi értékre belépő átvitelbitek közötti kizáró VAGY művelet eredménye.

AZ INTEL 8088 CÍMZÉSI MÓDJAI

Közvetlen adat az utasításban (immediate addressing)

```
        MOV  AX,7
KONST   EQU  0FFFEH
        MOV  AX,KONST
```

Regisztercímzés

```
        MOV  AX,CX
        MOV  DL,AH
```

Direkt címzés

	alapértelmezés szerinti szegmensregiszter
MOV AX,MEM_SZO	DS
MOV MEM_BYTE,BL	DS

Indirekt regiszteres címzés

MOV AX,[BX]	DS
MOV AX,[BP]	SS
MOV [SI],AX	DS
MOV [DI],AX	DS

Indexelt címzés

MOV MEM_SZO[BX],AX	DS
MOV DI,MEM_SZO[BX][SI]	DS
MOV CL,MEM_BYTE[DI]	DS
MOV AX,[BX][SI]	DS
MOV AX,[BX][DI]	DS
MOV AX,[BP][SI]	SS
MOV AX,[BP][DI]	SS

Megjegyzés

Egy utasításban csak egy memóriaoperandus lehet! Egy memóriaoperandus megadásánál legfeljebb egy eltolási érték (displacement), ill. változónév, egy bázisregiszter (BX vagy BP) és egy indexregiszter (SI vagy DI) szerepelhet.

Az utasítások funkció szerinti csoportosítása

ADATMOZGATÓ UTASÍTÁSOK

STACKMŰVELETEK

ARITMETIKAI UTASÍTÁSOK

Összeadás	ADD	26
	ADC	25
	DAA	30
	AAA	24
Kivonás	SUB	58
	SBB	55
	DAS	30
	AAS	25
Szorzás	MUL	47
	IMUL	33
	AAM	24
Osztás	DIV	31
	IDIV	32
	AAD	24
Konverzió	CBW	27
	CWD	30
Összehasonlítás	CMP	29
Egyoperandusos aritmetika	NEG	47
	INC	34
	DEC	31

LOGIKAI UTASÍTÁSOK

	AND	26
	OR	48
	XOR	60
	NOT	48
	TEST	58

FORGATÓ- ÉS LÉPTETŐUTASÍTÁSOK

STRINGKEZELŐ UTASÍTÁSOK

VEZÉRLÉSÁTADÓ UTASÍTÁSOK

CIKLUSSZERVEZŐ UTASÍTÁSOK

PROCESSZORVEZÉRLŐ UTASÍTÁSOK

Az Intel 8088 processzor utasításkészlete

AAA (*A*SCII *a*djust for *a*ddition)

Működése Két nem tömörített BCD (ASCII) szám összeadása után úgy módosítja az AL-ben levő eredményt, hogy az nem tömörített decimális érték legyen.

Példa **AAA ; az összeadás után**

Flagek Beállítva AF,CF,
nemdefiniált OF,PF,SF,ZF.

AAD (*A*SCII *a*djust for *d*ivision)

Működése Két nem tömörített BCD (ASCII) szám osztása előtt úgy módosítja az AX-ben levő osztandót, hogy az osztás hányadosa az AL-ben nem tömörített decimális érték legyen.

Példa **AAD ;az osztás előtt**

Flagek Beállítva PF,SF,ZF,
nemdefiniált AF,CF,OF.

AAM (*A*SCII *a*djust for *m*ultiply)

Működése Két nem tömörített BCD (ASCII) szám szorzása után úgy módosítja az AX-ben levő eredményt, hogy az nem tömörített decimális érték legyen.

Példa **AAM ;a szorzás után**

Flagek Beállítva PF,SF,ZF,
nemdefiniált AF,CF,OF.

AAS (*A*SCII *a*djust for *s*ubtraction)

Működése Két nem tömörített BCD (ASCII) szám kivonása után úgy helyesbíti az AL-ben levő eredményt, hogy az nem tömörített decimális érték legyen.

Példa
```
AAS ;a kivonás után
```

Flagek

Beállítva	AF,CF,
nemdefiniált	OF,PF,SF,ZF.

ADC (*Add* with *c*arry)

Működése A két operandus között bináris összeadást végez, és az összeghez hozzáadja a CF tartalmát. Az eredmény az első (cél-) operandusban keletkezik.

Példák
```
ADC AX,DI                ;regiszter–regiszter
ADC CH,BL

ADC DX,MEM_SZO           ;regiszter–memóriaope-
ADC NEV[SI],DI           ;randus
ADC CX,NEV[BX][SI]

ADC AL,3                 ;közvetlen adat
ADC BX,12H
ADC MEM_BYTE,4
```

Flagek Beállítva AF,CF,OF,PF,SF,ZF.

ADD (*Add*ition)

Működése A két operandus között bináris összeadást végez. Az eredmény az első (cél-) operandusban keletkezik.

Példák

```
ADD CX,DX              ;regiszter—regiszter
ADD BH,CL

ADD BX,MEM_SZO         ;regiszter—memóriaope-
ADD DX,NEV[BP][DI]     ;randus
ADD MEM_BYTE,BH

ADD AL,3               ;közvetlen adat
ADD MEM_SZO,48
```

Flagek Beállítva AF, CF, OF, PF, SF, ZF.

AND (Logical *AND*)

Működése A két operandus megfelelő bitjei között logikai ÉS műveletet hajt végre. Az eredmény az első (cél-) operandusban keletkezik.

Példák

```
AND AX,BX              ;regiszter—regiszter
AND BH,CL

AND SI,MEM_SZO         ;regiszter—memóriaope-
AND NEV[BX][DI],SI     ;randus

AND AL,7AH             ;közvetlen adat
AND MEM_BYTE,01010000B
```

Flagek Beállítva CF=0, OF=0, PF, SF, ZF,
nemdefiniált AF.

CALL (*Call* a procedure)

Működése A stack tetejére helyezi a következő utasítás címét (a visszatérési címet), és a vezérlést átadja az operandus által meghatározott memóriacímre. (Szegmensen belüli szubrutinhívásnál az IP tartalmát, szegmensek közötti hívásnál CS:IP tartalmát menti.)

Példák

```
CALL NEAR_PROC          ;szegmensen belüli direkt
                        ;hívás
CALL FAR_PROC           ;szegmensen kívüli direkt
                        ;hívás
CALL PROC_TABLA[SI]     ;szegmensen belüli indi-
                        ;rekt hívás
CALL MEM_DUPLA_SZO      ;szegmensen kívüli indi-
                        ;rekt hívás
CALL CX                 ;regiszteres indirekt hívás
```

Flagek Egyiket sem állítja.

CBW (*C*onvert *b*yte to *w*ord)

Működése Az AL regiszter tartalmának előjelét kiterjeszti az AH-ra.

Példa

```
CBW
```

Flagek Egyiket sem állítja.

CLC (*Cl*ear *c*arry flag)

Működése A CF-et 0-ra állítja.

Példa **CLC**

Flagek CF=0.

CLD (*Cl*ear *d*irection flag)

Működése A DF-et 0-ra állítja, így a stringműveletek a tárban a nagyobb címek felé haladva hajtódnak végre.

Példa **CLD**

Flagek DF=0.

CLI (*Cl*ear *i*nterrupt flag)

Működése Az IF-et 0-ra állítja, ezzel letiltja a processzor INTR vonalán érkező megszakításkéréseket, az ún. maszkolható külső megszakításokat.

Példa **CLI**

Flagek IF=0.

CMC (*Com*plement *c*arry flag)

Működése A CF állapotát az ellenkezőjére változtatja.

Példa **CMC**

Flagek Beállítva CF.

CMP (*Comp*are)

Működése Az első operandusból kivonja a másodikat, de csak a flageket állítja be az eredmény szerint, a két operandus változatlan marad.

Példák

```
CMP BX,CX          ;regiszter—regiszter
CMP BH,AL

CMP MEM_SZO,SI     ;regiszter—memóriaoperandus
CMP DI,NEV[BP][SI]

CMP AL,6           ;közvetlen adat
CMP MEM_BYTE,6AH
```

Flagek Beállítva AF,CF,OF,PF,SF,ZF.

CMPS/CMPSB/CMPSW (*Comp*are *s*tring)

Működése A DS:SI-vel címzett byte-os vagy szavas operandusból kivonja az ES:DI-vel címzett operandust, de úgy, hogy csak a flageket állítja. Ezután automatikusan növeli, ill. csökkenti az SI, DI string pointereket a DF állapota szerint. Ismétlési prefixummal látható el.

Példák

```
MOV SI,OFFSET STRING1
MOV DI,OFFSET STRING2
CMPS STRING1,STRING2

CMPSB

REPE CMPSW         ;ismétlési prefixummal
```

Flagek Beállítva AF,CF,OF,PF,SF,ZF.

CWD (*C*onvert *w*ord to *d*oubleword)

Működése Az AX regiszter tartalmának előjelét kiterjeszti a DX regiszterre. A DIV utasítással kapcsolatban használatos.

Példa **CWD**

Flagek Egyiket sem állítja.

DAA (*D*ecimal *a*djust for *a*ddition)

Működése Két tömörített decimális szám összeadása után úgy helyesbíti az AL-ben levő összeget, hogy az tömörített decimális érték legyen.

Példa **DAA ;az összeadás után**

Flagek Beállítva AF,CF,PF,SF,ZF,
nemdefiniált OF.

DAS (*D*ecimal *a*djust for *s*ubtraction)

Működése Két tömörített decimális szám kivonása után úgy helyesbíti az AL-ben levő eredményt, hogy az tömörített decimális érték legyen.

Példa **DAS ;a kivonás után**

Flagek Beállítva AF,CF,PF,SF,ZF,
nemdefiniált OF.

DEC (*Dec*rement by 1)

Működése Az operandusból levon egyet, és az eredményt ugyanabba az operandusba helyezi.

Példák
```
DEC BX
DEC AL
DEC MEM_BYTE
DEC MEM_SZO
DEC NEV[BX][SI]
```

Flagek	Beállítva	AF,OF,PF,SF,ZF. (CF-et nem módosítja!)

DIV (*Div*ision, unsigned)

Működése Előjel nélküli osztást hajt végre az AL és AH-ban (8 bites művelet), ill. az AX és DX-ben (16 bites művelet) tárolt osztandóval, valamint az utasítás operandusával. A hányadost az AL-ben, ill. az AX-ben kapjuk, a maradék az AH-ban, ill. a DX-ben keletkezik.

Példák
```
DIV CL
DIV BX
DIV MEM_BYTE
DIV MEM_SZO
DIV TABLA[SI]
```

Flagek	Nemdefiniált	AF,CF,OF,PF,SF,ZF.

ESC (*Esc*ape)

Működése Ez az utasítás lehetővé teszi, hogy multiprocesszoros környezetben egy másik processzor a 8088-tól kapjon utasítást végrehajtásra. Az első operandus a külső processzor műveleti kódja, a második operandust a 8088 az általa kezelt memóriából vagy regisztereiből küldi a másik processzorhoz.

Példák
ESC 6,MEM_BYTE
ESC 20,AL

Flagek Egyiket sem állítja.

HLT (*Halt*)

Működése Az utasítás hatására a processzor leállított állapotba kerül, amit külső megszakítás vagy egy reset jel szüntet meg.

Példa **HLT**

Flagek Egyiket sem állítja.

IDIV (*I*nteger *div*ision)

Működése Előjeles osztást hajt végre. Működése megegyezik a DIV utasításával. A maradék előjele megegyezik az osztandó előjelével.

Példák
IDIV CL
IDIV BX
IDIV MEM_BYTE
IDIV MEM_SZO
IDIV NEV[SI]

Flagek Nemdefiniált AF,CF,OF,PF,SF,ZF.

IMUL (*I*nteger *mul*tiplication)

Működése Előjeles szorzást hajt végre. Működése megegyezik a MUL utasításával.

Példák
```
IMUL DL
IMUL CX
IMUL MEM_BYTE
IMUL MEM_SZO
IMUL NEV[DI]
```

Flagek CF=1, OF=1, ha az eredmény felső fele nem az alsó fél előjel kiterjesztése,
nemdefiniált AF,PF,SF,ZF.

IN (*I*nput byte or word)

Működése Egy byte-ot (vagy szót) visz át a megadott input portról az AL (vagy AX) regiszterbe.

Példák
```
IN AX,SZO_PORT   ;0--255 közötti közvetlen adat-
IN AL,BYTE_PORT  ;ként megadott portszám

IN AX,DX         ;a port száma DX-ben
IN AL,DX         ;(max. 64 K)
```

Flagek Egyiket sem állítja.

INC (*Inc*rement destination by 1)

Működése A megadott operandushoz egyet hozzáad.

Példák
```
INC AX
INC DI
INC MEM_BYTE
INC NEV[BX][DI]
INC WORD PTR[BX]
INC BYTE PTR[SI]
```

Flagek Beállítva AF,OF,PF,SF,ZF.
(CF-et nem módosítja!)

INT (*Int*errupt)

Működése A flagszót a stackre helyezi (mint a PUSHF utasítás), törli a TF, IF flageket, és a vezérlést a 256 elemű megszakítási vektornak az operandus által kijelölt elemére adja, miután a visszatérési címet (CS,IP) már a stackre tette.

Példák
```
INT 3          ;egy byte-os utasítás
INT 21H        ;két byte-os utasítás
```

Flagek Beállítva TF=0, IF=0.

INTO (*Int*errupt if *o*verflow)

Működése Ha OF=1, megszakítást hajt végre a megszakítási vektor 4 elemére.

Példa
```
INTO
```

Flagek Beállítva TF=0, IF=0.

IRET (*I*nterrupt *ret*urn)

Működése A vezérlést visszaadja az előző megszakítási művelet által mentett visszatérési címre, és visszaállítja a kimentett flagszót.

Példa **IRET**

Flagek Az összes flaget visszatölti a stackről.

JA/JNBE (*J*ump if *a*bove/*J*ump if *n*ot *b*elow nor *e*qual)

Működése Ha CF=0, és ezzel egyidejűleg ZF=0, a megadott címkére adja a vezérlést (rövid ugrás, azaz előre max. 127 byte-ot, vissza max. 128 byte-ot ugorhat).

Példák **JA CEL_CIMKE**
JNBE CEL_CIMKE

Flagek Egyiket sem állítja.

JAE/JNB/JNC (*J*ump if *a*bove or *e*qual/*J*ump if *n*ot *b*elow/*J*ump if *n*ot *c*arry)

Működése Ha CF=0, a megadott címkére adja a vezérlést (rövid ugrás, azaz előre max. 127 byte-ot, vissza max. 128 byte-ot ugorhat).

Példák **JAE CEL_CIMKE**
JNB CEL_CIMKE
JNC CEL_CIMKE

Flagek Egyiket sem állítja.

JB/JNAE/JC (*J*ump if *b*elow/*J*ump if *n*ot *a*bove nor *e*qual/*J*ump if *c*arry)

Működése Ha CF=1, a megadott címkére adja a vezérlést (rövid ugrás, azaz előre max. 127 byte-ot, vissza max. 128 byte-ot ugorhat).

Példák
```
JB CEL_CIMKE
JNAE CEL_CIMKE
JC CEL_CIMKE
```

Flagek Egyiket sem állítja.

JBE/JNA (*J*ump if *b*elow or *e*qual/*J*ump if *n*ot *a*bove)

Működése Ha CF=1 vagy ZF=1, a megadott címkére adja a vezérlést (rövid ugrás, azaz előre max. 127 byte-ot, vissza max. 128 byte-ot ugorhat).

Példák
```
JBE CEL_CIMKE
JNA CEL_CIMKE
```

Flagek Egyiket sem állítja.

JCXZ (*J*ump if *CX* is *z*ero)

Működése Ha CX=0, a megadott címkére adja a vezérlést (rövid ugrás, azaz előre max. 127 byte-ot, vissza max. 128 byte-ot ugorhat).

Példa
```
JCXZ CEL_CIMKE
```

Flagek Egyiket sem állítja.

JE/JZ (*J*ump if *e*qual/*J*ump if *z*ero)

Működése Ha ZF=1, a megadott címkére adja a vezérlést (rövid ugrás, azaz előre max. 127 byte-ot, visszza max. 128 byte-ot ugorhat).

Példák **JE CEL_CIMKE**
JZ CEL_CIMKE

Flagek Egyiket sem állítja.

JG/JNLE (*J*ump if *g*reater/*J*ump if *n*ot /ess nor *e*qual)

Működése Ha ZF=0 és SF=OF, a megadott címkére adja a vezérlést (rövid ugrás, azaz előre max. 127 byte-ot, vissza max. 128 byte-ot ugorhat).

Példák **JG CEL_CIMKE**
JNLE CEL_CIMKE

Flagek Egyiket sem állítja.

JGE/JNL (*J*ump if *g*reater or *e*qual/*J*ump if *n*ot /ess)

Működése Ha SF=OF, a megadott címkére adja a vezérlést (rövid ugrás, azaz előre max. 127 byte-ot, vissza max. 128 byte-ot ugorhat).

Példák **JGE CEL_CIMKE**
JNL CEL_CIMKE

Flagek Egyiket sem áliítja.

JL/JNGE (*J*ump if *l*ess/*J*ump if *n*ot *g*reater nor *e*qual)

Működése Ha SF nem egyenlő OF-fel, a megadott címkére adja a vezérlést (rövid ugrás, azaz előre max. 127 byte-ot, vissza max. 128 byte-ot ugorhat).

Példák
```
JL CEL_CIMKE
JNGE CEL_CIMKE
```

Flagek Egyiket sem állítja.

JLE/JNG (*J*ump if *l*ess or *e*qual/*J*ump if *n*ot *g*reater)

Működése Ha ZF = 1 vagy SF nem egyenlő OF-fel, a megadott címkére adja a vezérlést (rövid ugrás, azaz előre max. 127 byte-ot, vissza max. 128 byte-ot ugorhat).

Példák
```
JLE CEL_CIMKE
JNG CEL_CIMKE
```

Flagek Egyiket sem állítja.

JMP (*Jump*)

Működése Feltétel nélküli vezérlésátadás az operandus által meghatározott címre.

Példák

```
JMP NEAR_CIMKE              ;szegmensen belüli
                            ;közeli ugrás

JMP CEL_CIMKE               ;szegmensen belüli
JMP SHORT NEAR_CIMKE        ;rövid ugrás

JMP FAR_CIMKE               ;szegmensek közötti
JMP FAR PTR NEAR_CIMKE      ;direkt ugrás

JMP VAR_DUPLASZO            ;szegmensek közötti
JMP DWORD PTR [BX][SI]      ;indirekt ugrás

JMP TABLA[BX]               ;szegmensen belüli
JMP WORD PTR [BX][SI]       ;indirekt ugrások;
JMP MEM_SZO                 ;az IP-t a megadott
JMP SI                      ;regiszter vagy
JMP BP                      ;memóriahely
                            ;tartalmával felülírják
```

Flagek Egyiket sem állítja.

JNE/JNZ (*Jump if not equal/Jump if not zero*)

Működése Ha ZF=0, a megadott címkére adja a vezérlést (rövid ugrás, azaz előre max. 127 byte-ot, visszza max. 128 byte-ot ugorhat).

Példák

```
JNE CEL_CIMKE
JNZ CEL_CIMKE
```

Flagek Egyiket sem állítja.

JNO (*J*ump if *n*ot *o*verflow)

Működése Ha OF=0, a megadott címkére adja a vezérlést (rövid ugrás, azaz előre max. 127 byte-ot, vissza max. 128 byte-ot ugorhat).

Példa **JNO CEL_CIMKE**

Flagek Egyiket sem állítja.

JNP/JPO (*J*ump if *n*ot *p*arity/*J*ump if *p*arity *o*dd)

Működése Ha PF=0, a megadott címkére adja a vezérlést (rövid ugrás, azaz előre max. 127 byte-ot, vissza max. 128 byte-ot ugorhat).

Példák **JNP CEL_CIMKE**
JPO CEL_CIMKE

Flagek Egyiket sem állítja.

JNS (*J*ump if *n*ot *s*ign)

Működése Ha SF=0, a megadott címkére adja a vezérlést (rövid ugrás, azaz előre max. 127 byte-ot, vissza max. 128 byte-ot ugorhat).

Példa **JNS CEL_CIMKE**

Flagek Egyiket sem állítja.

JO (*J*ump if *o*verflow)

Működése Ha OF=1, a megadott címkére adja a vezérlést (rövid ugrás, azaz előre max. 127 byte-ot, vissza max. 128 byte-ot ugorhat).

Példa **JO CEL_CIMKE**

Flagek Egyiket sem állítja.

JP/JPE (*J*ump if *p*arity/*J*ump if *p*arity *e*ven)

Működése Ha PF=1, a megadott címkére adja a vezérlést (rövid ugrás, azaz előre max. 127 byte-ot, vissza max. 128 byte-ot ugorhat).

Példák **JP CEL_CIMKE**
JPE CEL_CIMKE

Flagek Egyiket sem állítja.

JS (*J*ump if *s*ign)

Működése Ha SF=1, a megadott címkére adja a vezérlést (rövid ugrás, azaz előre max. 127 byte-ot, vissza max. 128 byte-ot ugorhat).

Példa **JS CEL_CIMKE**

Flagek Egyiket sem állítja.

LAHF (*L*oad *AH* from *f*lags)

Működése A flagszó alsó byte-ját (az SF,ZF,AF,PF és CF flageket) átviszi az AH regiszter megfelelő bitjeibe.

Példa **LAHF**

Flagek Egyiket sem állítja.

LDS (*L*oad pointer using *DS*)

Működése A duplaszó típusú második operandus szegmenscímrészét betölti a DS regiszterbe, offsetrészét pedig az első operandusként megadott regiszterbe.

Példák **LDS BX,DUPLA_SZO**
LDS SI,CIM_TABLA[BX]

Flagek Egyiket sem állítja.

LEA (*L*oad *e*ffective *a*ddress)

Működése A második operandus (memóriaoperandus) offsetcímét betölti az első operandusként megadott regiszterbe. (A memóriaoperandus indexelhető.)

Példák **LEA BX,HEXTAB**
LEA DX,[BX][SI]
LEA AX,NEV[BX][DI]

Flagek Egyiket sem állítja.

LES (*Load pointer using ES*)

Működése A duplaszó típusú második operandus szegmenscímrészét betölti az ES regiszterbe, offsetrészét pedig az első operandusként megadott regiszterbe.

Példák
```
LES BX,DUPLA_SZO
LES DI,CIM_TABLA[SI]
```

Flagek Egyiket sem állítja.

LOCK (*Lock* bus)

Működése Speciális 1-byte-os prefixum. Multiprocesszoros környezetben a processzor "bus-lock" jelet ad ki annak az utasításnak az idejére, amely előtt a LOCK prefixumot használtuk. Ezzel meggátolja, hogy a rendszer más processzorai hozzáférjenek a rendszer erőforrásaihoz.

Példa
```
LOCK XCHG FLAG,AL
```

Flagek Egyiket sem állítja.

LODS/LODSB/LODSW (*Load string*)

Működése A DS:SI által címzett stringből egy byte-ot (vagy szót) betölt az AL (vagy AX) regiszterbe, és a DF állapota szerint növeli, ill. csökkenti az SI string pointert.

Példák

```
LODS SZO_STRING

MOV SI,OFFSET BYTE_STRING
LODSB

MOV SI,OFFSET SZO_STRING
LODSW
```

Flagek Egyiket sem állítja.

LOOP (*Loop*)

Működése A CX tartalmát eggyel csökkenti, és ha még nem 0, a megadott címkére adja a vezérlést (csak rövid ugrás, azaz előre max. 127 byte-ot, vissza max. 128 byte-ot ugorhat).

Példa

```
LOOP CIMKE
```

Flagek Egyiket sem állítja.

LOOPE/LOOPZ (*Loop* if *e*qual/*Loop* if *z*ero)

Működése A CX tartalmát eggyel csökkenti, és ha CX nem 0 és ZF=1, a megadott címkére adja a vezérlést (csak rövid ugrás, azaz előre max. 127 byte-ot, vissza max. 128 byte-ot ugorhat).

Példák **LOOPE CIMKE**
LOOPZ CIMKE

Flagek Egyiket sem állítja.

LOOPNE/LOOPNZ (*Loop* if *n*ot *e*qual/*Loop* if *n*ot *z*ero)

Működése A CX tartalmát eggyel csökkenti, és ha CX nem 0 és ZF=0, a megadott címkére adja a vezérlést (csak rövid ugrás, azaz előre max. 127 byte-ot, vissza max. 128 byte-ot ugorhat).

Példák **LOOPNE CIMKE**
LOOPNZ CIMKE

Flagek Egyiket sem állítja.

MOV (*Move*)

Működése A második operandust az elsőbe mozgatja.

Példák

```
MOV CX,DX          ;regiszter–regiszter
MOV CL,DH

MOV MEM_BYTE,AL    ;regiszter–memóriaope-
MOV MEM_SZO,AX     ;randus
MOV AX,MEM_SZO
MOV DX,TOMB[SI]
MOV [BX][SI],DI

MOV ES,DX          ;szegmensregiszterekbe
MOV DS,AX
MOV SS,BX          ;CS-be nem lehet
MOV DS,SZEG_CIM

MOV DX,DS          ;szegmensregiszterekből
MOV AX,CS          ;ez engedélyezett
MOV MEM_SZO,SS

MOV AX,77          ;közvetlen adat
MOV CL,3
MOV MEM_SZO,0FFFFH
MOV BYTE PTR [DI],45
```

Flagek Egyiket sem állítja.

MOVS/MOVSB/MOVSW (*Move string*)

Működése A DS:SI által címzett byte-ot vagy szót átviszi az ES:DI által címzett byte-ra vagy szóra, majd a DF állapota szerint növeli, ill. csökkenti az SI és DI string pointereket. Ismétlési prefixummal látható el.

Példa

```
MOV SI,OFFSET STRING1
MOV DI,OFFSET STRING2
REP MOVSB
```

Flagek Egyiket sem állítja.

MUL (*Mu*ltiplication, unsigned)

Működése Előjel nélküli szorzást hajt végre az AL (vagy AX) tartalmával és a megadott 8 (vagy 16) bites operandussal. A keletkező szorzatot az AX-be (8 bites művelet), ill. a DX:AX-be (16 bites művelet) teszi.

Példák
```
MUL CL
MUL MEM_BYTE
MUL BX
MUL MEM_SZO
```

Flagek CF=1, OF=1, ha az eredmény felső része nem 0.

NEG (*Neg*ate)

Működése A megadott operandus 2-es komplemensét képzi.

Példák
```
NEG AL
NEG MEM_SZO
```

Flagek Beállítva AF, CF, OF, PF, SF, ZF.

NOP (*No op*eration)

Működése 3 órajel idejére a processzor semmilyen műveletet sem végez (időzítésre használható).

Példa
```
NOP
```

Flagek Egyiket sem állítja.

NOT (Logical *NOT*)

Működése Az operandus összes bitjét ellenkezőjére állítja (1-es komplemens).

Példák
```
NOT AH
NOT CX
NOT MEM_BYTE
```

Flagek Egyiket sem állítja.

OR (Logical inclusive *OR*)

Működése Logikai VAGY műveletet hajt végre a két operandus megfelelő bitjei között. Az eredmény az első operandusban keletkezik.

Példák
```
OR AH,BL                ;regiszter–regiszter
OR CX,DI

OR AX,MEM_SZO           ;regiszter–memóriaope-
OR CL,MEM_BYTE[SI]      ;randus
OR MEM_BYTE,DH

OR AL,11110110B         ;közvetlen adat
OR CL,0F6H
OR MEM_SZO[BX],23FAH
```

Flagek Beállítva CF=0, OF=0, PF, SF, ZF,
nemdefiniált AF.

OUT (*Out*put byte or word)

Működése Az AL (vagy AX) regiszterből egy byte-ot (vagy szót) átvisz az operandus által meghatározott output portra.

Példák

```
OUT 44,AX          ;0–255 közötti közvet-
OUT 7,AL           ;len adatként megadott
                   ;portszám

OUT DX,AL          ;a port száma DX-ben
OUT DX,AX          ;(max. 64 K)
```

Flagek Egyiket sem állítja.

POP (*Pop* word off stack)

Működése A stack SP-vel címzett tetejéről egy szót átvisz a megadott operandusba, és SP tartalmát 2-vel növeli.

Példák

```
POP CX
POP DS             ;CS nem megengedett
POP NEV[BX]        ;memóriaoperandusba
```

Flagek Egyiket sem állítja.

POPF (*Pop* flags off stack)

Működése A stack SP-vel címzett tetején levő szó megfelelő bitjeiből feltölti a flageket, és az SP tartalmát 2-vel növeli.

Példa

```
POPF
```

Flagek Mindegyik flaget állítja.

PUSH (*Push* word onto stack)

Működése Az SP tartalmát 2-vel csökkenti, majd a megadott operandust a stack SP-vel címzett tetejére helyezi.

Példák
```
PUSH AX
PUSH SI
PUSH ES                 ;CS is megengedett
PUSH NEV[BX][DI]        ;memóriaoperandusból
```

Flagek Egyiket sem állítja.

PUSHF (*Push f*lags onto stack)

Működése Az SP tartalmát 2-vel csökkenti, és a flagszót a stack SP-vel címzett tetejére helyezi.

Példa
```
PUSHF
```

Flagek Egyiket sem állítja.

RCL (*R*otate *l*eft through *c*arry)

Működése Az első operandust balra forgatja a carry (átvitel-) biten át. A forgatás lépésszámát a második operandus adja meg. A második operandus vagy 1, vagy CL. A CL tartalma a művelet után is megmarad.

Példák
```
RCL AH,1
RCL CX,1
RCL MEM_BYTE,1
RCL DH,CL           ;a lépésszám a CL-ben
RCL AX,CL
RCL MEM_SZO,CL
```

Flagek Beállítva CF, OF.

RCR (*R*otate *r*ight through *c*arry)

Működése Az első operandust jobbra forgatja a carry biten át. A forgatás lépésszámát a második operandus adja meg. A második operandus vagy 1, vagy CL. A CL tartalma a művelet után is megmarad.

Példák

```
RCR AL,1
RCR DX,1
RCR MEM_SZO,1
RCR BH,CL          ;a lépésszám a CL-ben
RCR AX,CL
RCR NEV[BX][DI],CL
```

Flagek Beállítva CF, OF.

REP (*Rep*eat string operation)

Működése A REP prefixum után megadott stringművelet mindaddig ismételten végrehajtódik, amíg CX tartalma 0 nem lesz. CX tartalma minden ismétlés után eggyel csökken.

Példák

```
REP MOVSW
REP STOSB
```

Flagek A REP után szereplő stringutasítások befolyásolják.

REPE/REPZ (*Rep*eat string operation while *e*qual/while *z*ero)

Működése A REPE/REPZ prefixum után megadott stringművelet mindaddig ismételten végrehajtódik, amíg ZF = 1 és CX nem 0. CX tartalma minden ismétlés után eggyel csökken.

Példák
REPE CMPS STRING1,STRING2
REPZ CMPSB

Flagek A REPE/REPZ után szereplő stringutasítások befolyásolják.

REPNE/REPNZ (*Rep*eat string operation while *n*ot *e*qual/*n*ot *z*ero)

Működése A REPNE/REPNZ prefixum után megadott stringművelet mindaddig ismételten végrehajtódik, amíg ZF = 0 és CX nem 0. CX tartalma minden ismétlés után eggyel csökken.

Példák
REPNE SCASW
REPNZ SCASB STRING

Flagek A REPNE/REPNZ után szereplő stringutasítások befolyásolják.

RET (*Ret*urn from procedure)

Működése A szubrutint meghívó CALL művelettel a stack tetejére tett visszatérési címre adja a vezérlést. SP tartalmát 2-vel (ill. FAR típusú szubrutinhívást követően 4-gyel) növeli. Ha az utasításban közvetlen konstans szerepel, ennek értékét az SP-hez adja, így „eldobja" a stacken levő paramétereket.

Példák

```
RET
RET 4          ;eldobja a 2 paraméterszót,
               ;amit a hívás előtt a stackre
               ;tett
```

Flagek Egyiket sem állítja.

ROL (*Ro*tate *l*eft)

Működése Az első operandust balra forgatja. A forgatás lépésszámát a második operandus adja meg. A második operandus vagy 1, vagy CL. A CL tartalma a művelet után is megmarad.

Példák

```
ROL BH,1
ROL CX,1
ROL NEV[DI],1
ROL DH,CL          ;a lépésszám a CL-ben
ROL MEM_BYTE,CL
```

Flagek Beállítva CF, OF.

ROR (*Ro*tate *r*ight)

Működése Az első operandust jobbra forgatja. A forgatás lépésszámát a második operandus adja meg. A második operandus vagy 1, vagy CL. A CL tartalma a művelet után is megmarad.

Példák
```
ROR CH,1
ROR AX,1
ROR MEM_BYTE,1
ROR DH,CL          ;a lépésszám a CL-ben
ROR AX,CL
ROR NEV[BX][DI],CL
```

Flagek Beállítva CF, OF.

SAHF (*S*tore *AH* into *f*lags)

Működése Az AH megfelelő bitjeit az SF, ZF, AF, PF és CF flagekbe mozgatja.

Példa
```
SAHF
```

Flagek Beállítva AF, CF, PF, SF, ZF.

SAL/SHL (*S*hift *a*rithmetic /eft/*Sh*ift /ogical /eft)

Működése Az első operandust balra lépteti. A lépésszámot a második operandus adja meg. A második operandus vagy 1, vagy CL. A CL tartalma a művelet után is megmarad. A legalacsonyabb helyi értékű bitre nullákat léptet be.

Példák
```
SAL AH,1
SHL CX,1
SAL MEM_SZO[DI], 1
SHL MEM_BYTE,CL    ;a lépésszám a CL-ben
```

Flagek Beállítva CF, OF, PF, SF, ZF,
nemdefiniált AF.

SAR (*Shift arithmetic right*)

Működése Az első operandust jobbra lépteti. A lépésszámot a második operandus adja meg. A második operandus vagy 1, vagy CL. A CL tartalma a művelet után is megmarad. A legmagasabb helyi értékű bitre az eredeti operandus előjelének megfelelő bitek lépnek be (előjelmegőrzés).

Példák

```
SAR AH,1
SAR BX,1
SAR MEM_BYTE,1

SAR DH,CL          ;a lépésszám a CL-ben
SAR NEV[BX][SI],CL
```

Flagek

Beállítva	CF, OF, PF, SF, ZF,
nemdefiniált	AF.

SBB (*Subtract with borrow*)

Működése A második operandust kivonja az elsőből. Ha CF=1, kivon egyet az eredményből. A végeredmény az első operandusban keletkezik.

Példák

```
SBB AX,BX          ;regiszter–regiszter
SBB CH,DL

SBB DX,MEM_SZO     ;regiszter–memória-
SBB MEM_BYTE[DI],CL ;operandus

SBB AX,4           ;közvetlen adat
SBB MEM_SZO[BX][DI],67
```

Flagek

Beállítva	AF, CF, OF, PF, SF, ZF.

SCAS/SCASB/SCASW (*Scan string*)

Működése Az ES:DI-vel címzett byte-ot (szót) kivonja az AL (vagy AX) tartalmából, de csak a flageket állítja. A DF állapota szerint növeli vagy csökkenti a DI string pointert. Ismétlési prefixummal látható el.

Példa

```
MOV DI,OFFSET STRING
MOV CX, HOSSZ
MOV AL,'$'          ;a '$' keresése adott stringben
REPNE SCASB
```

Flagek Beállítva AF, CF, OF, PF, SF, ZF.

SHR (*Shift logical right*)

Működése Az első operandust jobbra lépteti. A lépésszámot a második operandus adja meg. A második operandus vagy 1, vagy CL. A CL tartalma a művelet után is megmarad. A megmagasabb helyi értékű bitre nullákat léptet be.

Példák

```
SHR CX,1
SHR MEM_BYTE[DI],1
SHR DX,CL           ;a lépésszám a CL-ben
SHR NEV[BX][SI],CL
```

Flagek Beállítva CF, OF, PF, SF, ZF,
nemdefiniált AF.

STC (*Set carry flag*)

Működése A carry flaget 1-re állítja.

Példa

```
STC
```

Flagek Beállítva CF = 1.

STD (*Set d*irection flag)

Működése A DF flaget 1-re állítja.

Példa **STD**

Flagek Beállítva DF=1.

STI (*Set i*nterrupt flag)

Működése Az IF flaget 1-re állítja, ezzel a következő utasítás végrehajtása után engedélyezi a maszkolható külső megszakítások érvényre jutását.

Példa **STI**

Flagek Beállítva IF=1.

STOS/STOSB/STOSW (*Sto*re byte or word *s*tring)

Működése Az AL-ben levő byte-ot (vagy az AX-ben levő szót) az ES:DI-vel címzett byte-ba (vagy szóba) viszi. A DF állapotától függően növeli vagy csökkenti a DI string pointert. Ismétlési prefixummal látható el.

Példa
```
MOV DI,OFFSET STRING
STOSB
```

Flagek Egyiket sem állítja.

SUB (*Sub*traction)

Működése Az első operandusból kivonja a másodikat. Az eredmény az első operandusban keletkezik.

Példák

```
SUB AX,BX                    ;regiszter–regiszter
SUB CH,DL

SUB DX,MEM_SZO               ;regiszter–memó-
SUB MEM_BYTE[BX],CL          ;riaoperandus

SUB AL,6                     ;közvetlen adat
SUB MEM_SZO[BX][SI],1987
```

Flagek Beállítva AF, CF, OF, PF, SF, ZF.

TEST (*Test* or non-destructive logical AND)

Működése A két operandus megfelelő bitjei között logikai ÉS műveletet hajt végre, de csak a flageket állítja, az operandusok változatlanok maradnak.

Példák

```
TEST AX,DX                   ;regiszter–regiszter
TEST BH,CL

TEST MEM_SZO,SI              ;regiszter–memória-
TEST BL,MEM_BYTE[BX]         ;operandus

TEST AL,0FH                  ;közvetlen adat
TEST MEM_SZO,00FFH
```

Flagek Beállítva CF=0, OF=0, PF, SF, ZF,
nemdefiniált AF.

WAIT (*Wait* while TEST pin not asserted)

Működése A processzort várakozó állapotba lépteti, ha a TEST bemenetén nincs jel. A processzort ebből az állapotából a TEST jel megérkezése léptetheti ki. A várakozó állapotban egy külső megszakítás érvényre juthat, de utána a processzor ismét visszakerül a várakozó állapotba.

Példa

```
WAIT
```

Flagek Egyiket sem állítja.

XCHG (Exchange)

Működése A két operandus tartalmát felcseréli. (Szegmensregiszter nem lehet operandus.)

Példák

```
XCHG AX,BX          ;regiszter—regiszter
XCHG SI,AX
XCHG BL,AH

XCHG MEM_BYTE,CL    ;regiszter—memória-
XCHG DX,MEM_SZO     ;operandus
```

Flagek Egyiket sem állítja.

XLAT (Translate)

Működése Keresést hajt végre egy táblázatban. Az AL regisztert a táblázat indexeként használja. A táblázat elejét a BX regiszterrel kell címezni. A táblázatban megtalált byte-ot az AL-ben adja vissza.

Példa

```
MOV BX,OFFSET HEXTAB
MOV AL,ERTEK
XLAT
```

Flagek Egyiket sem állítja.

XOR (Logical exclusive *OR*)

Működése A két operandus megfelelő bitjei között logikai kizáró VAGY műveletet hajt végre. Az eredmény az első operandusba kerül.

Példák

```
XOR AH,BL                    ;regiszter–regiszter
XOR CX,DI

XOR CL,MEM_BYTE              ;regiszter–memória-
XOR MEM_SZO[BX][DI],DX       ;operandus

XOR AL,11110110B             ;közvetlen adat
XOR DI,23F5H
XOR NEV[BX],67
```

Flagek

Beállítva	CF=0, OF=0, PF, SF, ZF,
nemdefiniált	AF.

Direktívák (pszeudoutasítások)

ADATOKKAL KAPCSOLATOS DIREKTÍVÁK

ASSUME szegmensregiszter:szegmensnév vagy
NOTHING [...]

Célja — Közli az assemblerrel, hogy a futási időben az adott szegmens utasításai, ill. adatai, mely szegmensregisztereken keresztül lesznek címezhetők. Az assembler a szegmensen belüli relatív címeket ezzel a feltételezéssel számítja ki. A szegmensregiszterek betöltése a rendszerbetöltő program vagy a programozó feladata.

Példa

```
ASSUME DS:ADAT,SS:STACK,CS:PROG,
       ES:NOTHING
```

COMMENT határoló_karakter szöveg határoló_karakter

Célja — A forrásprogramba megjegyzés írható a ,;' jel használata nélkül.

Példa

```
COMMENT + A megjegyzések
*
*
akárhány sorba írhatók! +
```

változónév **DB** kifejezés

Célja Változót definiál vagy tárterületet inicializál. Egy vagy több byte-ot foglal le.

Példák

```
SZAM DB 27
BYTE DB ?                          ;csak területfog-
                                   ;lalás
SZOVEG DB 'ABCDEFG HI'

ISM DB 10 DUP(?)                   ;10 byte-os tömb
                                   ;lefoglalása
MOV AL,ISM[5]                      ;hivatkozás a
                                   ;tömb 6. byte-jára

TABLA DB 100 DUP (5 DUP(4) ,7)     ;100-szor
                                   ;4,4,4,4,4,7
SZAMSOR DB 1,2,3,0FH,0BFH
```

változónév **DD** kifejezés

Célja Változót definiál vagy tárterületet inicializál. Két szót (4 byte-ot) foglal le.

Példák

```
CIMKONST DD TABLA ;a TABLA változó 16 bites
                  ;OFFSET és 16 bites SEG
                  ;értéke
LISTA DD 'AB',2 DUP(?) ;42H, 41H, 00H, 00H és
                       ;még két duplaszónyi hely
```

változónév **DQ** kifejezés

Célja Változót definiál vagy tárterületet inicializál. 4 szót (8 byte-ot) foglal le.

Példák

```
PI DQ 3141597
STRING DQ 'AB'          ;max 2 karakter
```

változónév **DT** kifejezés

Célja Változót definiál vagy tárterületet inicializál. 10 byte hosszúságú tömörített decimális értéket hoz létre.

Példák
```
PAKOLT_DEC DT 1234567890
DEC_VALT DT ?
```

változónév **DW** kifejezés

Célja Változót definiál vagy tárterületet inicializál. Egy szót (2 byte-ot) foglal le.

Példák
```
VALT DW 0FFF0H
CIMKONST DW TABLA ;a TABLA változó 16 bites
                  ;OFFSET értéke
SOROZAT DW 2 DUP(2,3(DUP(1))) ;2-szer 2,1,1,1
STACK DW 100H DUP(?)
```

END [kifejezés]

Célja A forrásprogram végét jelzi. A „kifejezés" a forrásprogram kezdőcímét adja meg.

Példa
```
END KEZD
```

név **EQU** kifejezés

Célja A „kifejezés" értékét hozzárendeli a „név"-hez. Konstans értékadás, amely a forrásmodul egészére vonatkozik.

Példák
```
KONST EQU 7
B EQU [BP+8]
CBD EQU AAD
```

címke = kifejezés

Célja Az „=" direktíva olyan konstansérték-adás, amely (szemben az EQU-val) a forrásmodulban bármikor újradefiniálható.

Példák
```
KONST = 8
...
KONST = KONST + 1
```

EVEN

Célja Hatására a helyszámláló páros címhatárra áll (egy NOP beszúrásával, ha szükséges).

Példa
```
EVEN
```

EXTRN név:típus [...]

Célja Az adott modulban felhasznált, de egy külső modulban definiált (és ott PUBLIC-kal megjelölt) szimbólumokat adja meg a fordításhoz és a szerkesztéshez. A „típus" **BYTE, WORD, DWORD, QWORD, TBYTE, NEAR, FAR** és **ABS** lehet.

Példa
```
EXTRN CIMKE:FAR
```

név **GROUP** szegmensnév [...]

Célja Előírja, hogy a felsorolt nevű szegmensek mind ugyanazon a 64 kbyte-os tárterületen belül legyenek. Az offsetcímeket az assembler a GROUP elejétől számolja.

Példa
```
DGROUP GROUP SEG1,SEG2
SEG1 SEGMENT
     ASSUME DS:DGROUP
     ...
SEG1 ENDS
SEG2 SEGMENT
     ASSUME DS:DGROUP
     ...
SEG2 ENDS
```

INCLUDE filespec

Célja Fordításkor a forrásprogram adott helyére másolja a megadott másik forrásfile-t.

Példa
```
INCLUDE B:SAJATFOR.ASM
```

név **LABEL** típus

Célja A „név" típusát definiálja. A „típus" adatterület esetén **BYTE, WORD, DWORD, QWORD, TBYTE,** struktúranév és rekordnév, kód esetén **NEAR** és **FAR** lehet.

Példák
```
BYTE_TOMB LABEL BYTE
SZO_TOMB DW 100 DUP(0)

TAV_RUTIN LABEL FAR
```

NAME modulnév

Célja Nevet ad a fordítás során keletkező tárgymodulnak. Ha elmarad a NAME direktíva, akkor a forrásfile neve lesz a modulnév.

Példa
```
NAME PROG1
```

ORG kifejezés

Célja Az assembler a helyszámlálót a „kifejezés" értékére állítja.

Példák
```
ORG 100H
ORG $+2          ;2 byte-ot előreugrik
```

eljárásnév **PROC** [**NEAR**] (vagy **PROC FAR**)
...
RET
...
eljárásnév **ENDP**

Célja Egy külön funkciót ellátó utasításcsoportot (szubrutint) azonosít, és megadja az eljárásban szereplő RET utasítások típusát.

PUBLIC szimbólum [...]

Célja Az adott modulban definiált szimbólumot hozzáférhetővé teszi a külső modulok számára.

Példa
```
PUBLIC FAR_RUTIN
```

RADIX kifejezés

Célja Az alapértelmezés szerinti decimális konstansértelmezést tetszőleges alapú számrendszerre állítja (2 és 16 között).

Példa
```
MOV BX,0FFFH
.RADIX 16
MOV BX,0FFF
```

rekordnév **RECORD** mezőnév:szélesség[=kifejezés],
[...]

Célja A rekord 1 vagy 2 byte-os bitsorozat, amelyen belül bitmezőkre hivatkozhatunk. A tárterület tényleges lefoglalására a „rekordnév" használható direktívaként. A „mezőnevek" konstansok, amelyek értékei megegyeznek a mezőnek a rekord jobb szélső pozíciójától mért távolságával. A „kifejezés"-sel a rekordmezőnek kezdeti érték adható.

Példa

```
ABC RECORD R:7,E:3,D:6
...
AKTIV ABC <6,5,3>     ;tárfoglalás inicializálással
...
MOV DX,AKTIV          ;az E mező elkülönítése
AND DX,MASK E         ;az AKTIV rekordban
MOV CL,E
SHR DX,CL
```

szegmensnév **SEGMENT** [illesztési_típus]
[kombinálhatósági_típus]
['osztály']
...
szegmensnév **ENDS**

Célja szegmensdefiniálás.

Az illesztési_típus	**PARA** (alapértelmezés), **BYTE**, **WORD**, **PAGE**.
A kombinálhatósági_típus	**PUBLIC** (alapértelmezés), **COMMON**, **AT** kifejezés, **STACK**, **MEMORY**.
Az 'osztály'	tetszőleges.

struktúranév **STRUC**

...

változónév **DB vagy DW vagy DD vagy DT vagy DQ** kifejezés

...

struktúranév **ENDS**

Célja Az adatstruktúra definiálása.

Példa

```
ABC STRUC
MEZO1 DB 1,2
MEZO2 DB 10 DUP(?)
MEZO3 DB 5
ABC ENDS
...
TER ABC <,,7>          ;területfoglalás és újrainicia-
                       ;lizálás
...
MOV AL,TER.MEZO3       ;hivatkozás a struktúra egyik
                       ;elemére
MOV BX,OFFSET TER      ;a hivatkozás másik módja
MOV AL,MEZO1.[BX]
```

IFxx argumentum
...
[**ELSE**]
...
ENDIF

Célja A feltételesen befordítandó utasításblokk definiálása. Ha az IFxx által megfogalmazott feltétel igaz, az ELSE-ig terjedő utasításblokk fordítódik be, egyébként az ELSE és az ENDIF közötti. Ha nincs ELSE ág, akkor a feltétel teljesülése esetén az egész ENDIF-ig terjedő rész befordul.

Változatok

IF kifejezés
Igaz, ha a kifejezés nem 0.

IFE kifejezés
Igaz, ha a kifejezés=0.

IF1
Igaz, ha az első fordítási menet folyik.

IF2
Igaz, ha a második fordítási menet folyik.

IFDEF szimbólum
Igaz, ha a szimbólum definiált, vagy EXTRN-nel külső szimbólumnak deklaráltuk.

IFNDEF szimbólum
Az IFDEF ellentettje.

IFB <argumentum>
Igaz, ha az argumentum blank.

IFBN <argumentum>
Az IFB ellentettje.

IFIDN <arg1> <arg2>
Igaz, ha a string típusú arg1 azonos arg2-vel.

IFDIF <arg1> <arg2>
Igaz, ha a string típusú arg1 különbözik az arg2-től.

ENDIF
Az IFxx direktíva lezárása.

ELSE
Ha az IFxx feltétele nem teljesül, az ELSE utáni utasítások fordítódnak.

név **MACRO** paraméterlista

...

...

ENDM

Célja Egy makró definiálása. A makró a „név"-vel aktiválható.

Példa

```
COPY MACRO FROM,TO,SIZE
     MOV SI,OFFSET FROM
     MOV DI,OFFSET TO
     MOV CX,SIZE
     CALL COPYSTRING
     ENDM
     ...
     ...
     COPY STR1,STR2,30          ;aktiválás
     ...
     COPY VALT1,VALT2,HOSSZ ;aktiválás
```

PURGE makrónév [...]

Célja Törli a megadott makrók definícióját. Ezzel a fordító tárterületének egyes részeit felszabadítja.

Példa **PURGE COPY**

LOCAL paraméterlista

Célja Hatására az assembler a makró törzsében szereplő szimbólumokat egyedivé teszi, így elkerülhető a többszöri definíció.

Példa

```
ABC MACRO XX
    LOCAL CIMKE1
    ...
CIMKE1: utasítás
        ...
        ENDM
```

REPT kifejezés

...

ENDM

Célja A blokkban levő utasítások a kifejtéskor „kifejezés"-szer ismétlődnek.

Példa

```
X=0
        REPT 10
X=X+1
        DB X    ;DB 1, DB 2 ... DB 10 kerül be a
                ;kifejtéskor
        ENDM
```

IRP paraméter,<argumentumlista>

...

ENDM

Célja Az utasításblokk annyiszor ismétlődik a kódban, ahány eleme van az „argumentumlistának". A „paraméter"-t minden ismétléskor a lista soron következő eleme helyettesíti minden olyan helyen, ahol a „paraméter" a blokkban előfordul.

Példa

```
IRP X,<3,5,7,9,11>
DB X            ;DB 3, DB 5 ... DB 11
ENDM
```

IRPC paraméter,string (vagy <string>)

...

ENDM

Célja Az utasításblokk a „string" minden egyes karakterére ismétlődik. A „paraméter"-t minden ismétléskor a „string" soron következő karaktere helyettesíti.

Példa

```
IRPC X,123456789
DB 'X'          ;DB 1, DB 2, ..., DB 9
ENDM
```

ENDM

Célja A MACRO, REPT, IRP és IRPC direktívákat zárja le.

Példa

```
ENDM
```

EXITM

Célja A makrokifejtés az EXITM direktívánál abbamarad. Feltételes direktívákkal együtt használatos.

Példa

```
ABC  MACRO X,Y
     ...
     ...
     IF kifejezés
     ...
     EXITM
     ENDIF
     ... ;ha a kifejezés nem 0, ezek az utasítások
     ... ;már nem kerülnek be a kifejtésbe
     ENDM
```

szöveg&szöveg

Célja Az „&" operátor segítségével a makró kifejtésénél szöveget vagy szimbólumokat lehet konkatenálni (összekapcsolni).

Példa
```
IRPC CHAR,ABCD
ADD AX,CHAR&X
ENDM
```

Kifejtve:
```
ADD AX,AX
ADD AX,BX
ADD AX,CX
ADD AX,DX
```

;;szöveg

Célja A „;;" operátort követő megjegyzés nem kerül bele a makró kifejtésébe.

Példa
```
PELDA  MACRO PAR
       DB 'MSG&PAR'  ;;ez a megjegyzés csak a
                     ;;definícióban szerepel
       ENDM
```

! karakter

Célja Használatával elérjük, hogy a speciális makro-operátorok egyszerű karakterként értelmeződjenek, és ne fejtsék ki hatásukat.

Példa

```
COPY MACRO HONNAN,HOVA,HOSSZ
     MOV SI,HONNAN
     MOV DI,HOVA
     MOV CX,HOSSZ
     CALL COPYSTRING
     ENDM
```

Aktiválás

```
COPY OFFSET! VALT1,OFFSET! VALT2,9
```

Kifejtés

```
MOV SI,OFFSET VALT1 ;a ! hatására a szó-
MOV DI,OFFSET VALT2 ;közt nem veszi para-
MOV CX,9            ;méterelválasztó ka-
                    ;rakternek
```

%kifejezés

Célja A kifejezést az aktuális alapszám szerinti számrendszerben numerikus értékké alakítja. A makró kifejtésekor számozni lehet vele.

Példa

```
ERTEK EQU 3
...
      PELDA %ERTEK
```

Kifejtés (a PELDA definícióját l.'a ;; operátornál):

```
DB 'MSG3'
```

.CREF

Célja Engedélyezi a kereszthivatkozási (cross reference) listakészítést. Ez az alapértelmezés.

.XCREF

Célja Letiltja a kereszthivatkozási listakészítést.

.LALL

Célja Makrokifejtéskor a makró teljes szövegét listázza.

.SALL

Célja Letiltja a makrók által generált szöveg- és tárgykód listázását.

.XALL

Célja Csak azokat a forrássorokat listázza, amelyekből a makró kifejtésekor tárgykód generálódik. Ez az alapértelmezés.

.LIST

Célja Engedélyezi a forrás- és tárgykódú listakészítést. Ez az alapértelmezés.

.XLIST

Célja Letiltja a forrás- és tárgykódú listakészítést.

%OUT szöveg

Célja Fordítás közben kiírja a szöveget, ezzel nyomon követhetjük, hol tart az assembler a fordítással.

PAGE [op1] [op2]

Célja Operandusok nélkül: lapot dob, és a lapszámozást eggyel növeli.
„op1": az egy oldalra kerülő sorok számát adja meg 10 és 255 között. Az alapértelmezés 66. Ha az „op1" helyén + jel áll, a fejezet sorszámát eggyel növeli, és a lapszám 1-re áll.
„op2": a lap szélességét állítja be 60 és 132 karakter között. Az alapértelmezés 80.

TITLE szöveg

Célja Megadja azt a fejlécszöveget, amely minden lap első sorára kinyomtatódik.

SUBTTL szöveg

Célja Megadja azt a szöveget, amely minden lapra, a TITLE által megadott fejlécsor alá nyomtatódik.

.LFCOND

Célja Hatására a fordító minden feltételes blokkot listáz (azokat is, amelyek feltételei nem teljesülnek).

.SFCOND

Célja Letiltja a nem teljesülő feltételek blokkjainak listázását.

.TFCOND

Célja Vált az előző két listázási mód között.

Irodalom

Macro Assembler Reference Manual 3.0 (Microsoft Corporation)

Kiadja a Műszaki Könyvkiadó
Felelős kiadó: Szűcs Péter igazgató
Felelős szerkesztő: Györke Tiborné

A szedés a Műszaki Könyvkiadóban készült

AGROÉPSZER

Műszaki vezető: Kőrizs Károly
Műszaki szerkesztő: Molnár Zoltán
A borítót tervezte: Székely Edith
A könyv formátuma: AK/40
Ivterjedelme: 3,9 (A5)
Azonossági szám: 80026
MŰ: 4122–i–8790
Készült az MSZ 5601 és 5602 szerint
A kézirat lezárva: 1987. október

150 Ft

MŰSZAKI KÖNYVKIADÓ

NOVOTRADE RT