8086 KOMUT KÜMESİ VE ÖRNEKLER

Doç.Dr. M.Ali Akcayol

Gazi Üniversitesi
Bilgisayar Mühendisliği Bölümü

(2007)

NOT: Bu doküman başta emu8086 programının yardım dosyası olmak üzere aşağıdaki kaynaklar kullanılarak hazırlanmıştır:

- Barry B. Brey, The Intel Microprocessors 8086/8088, 80186/80188, 80286, 80386, 80486, Pentium, Pentium Pro Processor, Pentium II, Pentium III, Pentium 4 Architecture, Programming, and Interfacing (7th edition), Prentice Hall, ISBN: 0131974076, 2006.
- Walter A. Triebel, Avtar Singh, Avtar Singh, The 8088 and 8086 Microprocessors: Programming, Interfacing, Software, Hardware, and Applications (4th edition), Prentice Hall, 0130930814, 2002.
- Muhammad Ali Mazidi, Janice Gillispie-Mazidi, Muhammad A. Mazidi, Janice Catherine Gillispie-Mazidi, 80X86 IBM PC and Compatible Computers: Assembly Language, Design, and Interfacing (4th edition), Prentice Hall, 2002.

GENEL BİLGİLER

GENEL AMAÇLI REGISTER'LAR

AX, BX, CX, DX, SI, DI, BP, SP

ÖZEL AMAÇLI REGISTER'LAR

IP, FLAG REGISTER'LARI

SEGMENT REGISTER'LARI

CS, DS, ES, SS

OPERAND TÜRLERİ:

REG : AX, BX, CX, DX, AH, AL, BL, BH, CH, CL, DH, DL, DI, SI, BP, SP.
SREG : DS, ES, SS, CS(sadece ikinci operand).
memory : [BX], [BX+SI+7], degisken, vb.
immediate : 5, -24, 3Fh, 10001101b, vb.

BAYRAK DURUMLARI:

1 - komut bu bayrak bitini **1** yapar.
0 - komut bu bayrak bitini **0** yapar.
r - bayrak değeri komutun sonucuna bağlıdır.
? - bayrak değeri tanımsızdır (**1** veya **0** olabilir).

DEĞİŞKENLER

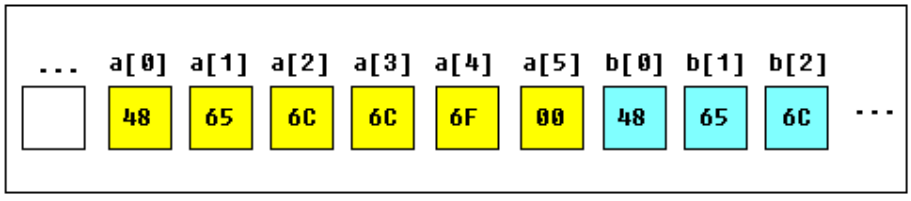
isim **DB** değer
isim **DW** değer
isim **DD** değer

Örnek:

```
ORG 100h

MOV AL, var1
MOV BX, var2

RET    ; program durur

VAR1 DB 7
var2 DW 1234h
```

DİZİLER

```
a DB 48h, 65h, 6Ch, 6Ch, 6Fh, 00h
b DB 'Hello', 0
```



```
MOV AL, a[3]
MOV SI, 3
MOV AL, a[SI]
```

DUP operatörü

adet **DUP** (değerler)

```
c DB 5 DUP(9)
c DB 9, 9, 9, 9, 9

d DB 5 DUP(1, 2)
d DB 1, 2, 1, 2, 1, 2, 1, 2, 1, 2
```

DEĞİŞKENLER

isim **EQU** <ifade>

```
k EQU 5
MOV AX, k

MOV AX, 5
```

BAYRAK BİTLERİ

C – Carry Flag
Z – Zero Flag
S – Sign Flag
O – Overflow Flag
P – Parity Flag
A – Auxiliary Flag
I - Interrupt Flag
D - Direction Flag

VERİ TRANSFER KOMUTLARI

PUSH, POP, LEA, LDS, LODS, LODSB, LODSW, STOS, STOSB, STOSW, MOVS, MOVSB, MOVSW, INS, INSB, INSW, OUTS, OUTSB, OUTSW, XCHG, LAHF, SAHF, XLAT, IN, OUT

ARİTMETİK VE MANTIK KOMUTLARI

ADD, INC, ADC, SUB, DEC, SBB, CMP, MUL, IMULİ DIV, IDIV, CBW, CWD, DAA, DAS, AAA, AAD, AAM, AAS, AND, OR, XOR, TEST, NOT, NEG, SHL, SHR, SAL, SAR, ROL, RCL, RCR, ROR, SCAS, CMPS

PROGRAM KONTROL KOMUTLARI

JMP, JA, JAE, JB, JBE, JC, JE, JZ, JG, JGE, JL, JLE, JNC, JNE, JNZ, JNC, JNE, JNZ, JNO, JNS, JNP, JPO, JO, JP, JPE, JS, JCXZ, LOOP, CALL, RET, INT, IRET, HLT, NOP

GENEL BİLGİLER

KOMUT	OPERANDLAR	TANIM VE ÖRNEK

AAA	-	Toplama sonrasında ASCII düzenlemesi yapar.
		BCD kodlarıyla çalışırken toplama sonrasında AH ve AL değerlerini düzenler.
		
		Algoritma:
		`if low nibble of AL > 9 or AF = 1 then`
		`  AL = AL + 6`
		`  AH = AH + 1`
		`  AF = 1`
		`  CF = 1`
		`else`
		`  AF = 0`
		`  CF = 0`
		
		İki durumdada AL'nin soldaki dörtlüsü sıfırlanır.
		
		Örnek:
		`MOV AX, 15  ; AH = 00, AL = 0Fh`
		`AAA  ; AH = 01, AL = 05`
		`RET`
		
		C Z S O P A
		r ? ? ? ? r

AAD	-	Bölme sonrasında ASCII düzenlemesi yapar.
		İki BCD değerini bölme için hazırlar.
		
		Algoritma:
		`AL = (AH * 10) + AL`
		`AH = 0`
		
		Örnek:
		`MOV AX, 0105h  ; AH = 01, AL = 05`
		`AAD  ; AH = 00, AL = 0Fh (15)`
		`RET`
		
		C Z S O P A
		? r r ? r ?

<table>
<tr>
<td>AAM</td>
<td>-</td>
<td>Çarpma sonrasında ASCII düzenlemesi yapar.
İki BCD değerinin çarpma sonucunu düzenler.

Algoritma:
<pre>AH = AL / 10
AL = remainder</pre>Örnek:
<pre>MOV AL, 15 ; AL = 0Fh
AAM ; AH = 01, AL = 05
RET</pre><table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td>?</td><td>r</td><td>r</td><td>?</td><td>r</td><td>?</td></tr></table></td>
</tr>
<tr>
<td>AAS</td>
<td>-</td>
<td>Çıkarma sonrasında ASCII düzenlemesi yapar.
BCD kodlarıyla çalışırken çıkarma sonrasında AH ve AL değerlerini düzenler.

Algoritma:
<pre>if low nibble of AL > 9 or AF = 1 then
      AL = AL - 6
      AH = AH - 1
      AF = 1
      CF = 1
else
      AF = 0
      CF = 0</pre>İki durumdada AL’nin soldaki dörtlüsü sıfırlanır.

Örnek:
<pre>MOV AX, 02FFh ; AH = 02, AL = 0FFh
AAS ; AH = 01, AL = 09
RET</pre><table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td>r</td><td>?</td><td>?</td><td>?</td><td>?</td><td>r</td></tr></table></td>
</tr>
<tr>
<td>ADC</td>
<td>REG, memory
memory, REG
REG, REG
memory, immediate
REG, immediate</td>
<td>Toplama yapar.

Algoritma:
<pre>operand1 = operand1 + operand2</pre>Örnek:
<pre>MOV AL, 5 ; AL = 5
ADD AL, -3 ; AL = 2
RET</pre><table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td></tr></table></td>
</tr>
</table>

<table>
<tr>
<td>AND</td>
<td>REG, memory
memory, REG
REG, REG
memory, immediate
REG, immediate</td>
<td>İki operand arasında karşılıklı tüm bitlere mantıksal VE işlemi yapar ve sonucu soldaki operanda kaydeder.

Doğruluk tablosu:

```
1 AND 1 = 1
1 AND 0 = 0
0 AND 1 = 0
0 AND 0 = 0
```


Örnek:

```
MOV AL, 'a' ; AL = 01100001b
AND AL, 11011111b ; AL = 01000001b ('A')
RET
```


<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td></tr><tr><td>0</td><td>r</td><td>r</td><td>0</td><td>r</td></tr></table></td>
</tr>
<tr>
<td>CALL</td>
<td>procedure name
etiket
4-byte adres</td>
<td>Bir prosedürü çağırır ve dönüş adresini (IP) stack’a push eder. 4 byte adres kullanılarak çağırma yapılabilir. 1234h:5678h adresinde ilk değer segment ikinci değer offset adresini belirtir. Bu far çağırmadır ve IP ile birlikte CS stack’a push edilir.

Örnek:

```
#make_COM#
ORG 100h    ; for COM file.
CALL p1
ADD AX, 1
RET         ; return to DOS.

p1 PROC     ; procedure declaration.
    MOV AX, 1234h
    RET     ; return to caller.
p1 ENDP
```


<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td colspan="6">değişmez</td></tr></table></td>
</tr>
<tr>
<td>CBW</td>
<td>-</td>
<td>Byte değeri word’e çevirir.

Algoritma:

```
if high bit of AL = 1 then
    AH = 255 (0FFh)
else
    AH = 0
```


Örnek:

```
MOV AX, 0   ; AH = 0, AL = 0
MOV AL, -5  ; AX = 000FBh (251)
CBW         ; AX = 0FFFBh (-5)
RET
```


<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td colspan="6">değişmez</td></tr></table></td>
</tr>
</table>

CLC	-	Taşma bayrağı (Carry flag) sıfırlanır. Algoritma: `CF = 0` C 0
CLD	-	Direction bayrağı (Direction flag) sıfırlanır. SI ve DI register'ları CMPSB, CMPSW, LODSB, LODSW, MOVSB, MOVSW, STOSB, STOSW komutlarıyla ardarda yapılan işlemlerde artırılır. Algoritma: `DF = 0` D 0
CLI	-	Interrup etkinleştirme bayrağı (Interrupt enable flag) sıfırlanır. Bu donanım interrupt'larını etkisiz yapar. Algoritma: `IF = 0` I 0

CMC	-	Taşma bayrağı (Carry flag) terslenir(complement).
		
		Algoritma:
		`if CF = 1 then CF = 0`
		`if CF = 0 then CF = 1`
		
		C
		R

<table>
<tr>
<td>CMP</td>
<td>REG, memory
memory, REG
REG, REG
memory, immediate
REG, immediate</td>
<td>Karşılaştırma yapar. Soldaki operand’ı sağdakinden çıkartır ancak operandlar değişmez sadece bayrak bitleri değişir.

Algoritma:
<code>operand1 - operand2</code>

OF, SF, ZF, AF,PF, CF bayrakları sonuca göre yeniden düzenlenir.

Örnek:
<pre>MOV AL, 5
MOV BL, 5
CMP AL, BL ; AL=5, ZF=1 (eşit)
RET</pre>
<table>
<tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr>
<tr><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td></tr>
</table>
</td>
</tr>
<tr>
<td>CMPSB</td>
<td>-</td>
<td>ES:[DI] ile DS:[SI] arasında bir byte karşılatırır.

Algoritma:
<code>DS:[SI] - ES:[DI]</code>

OF, SF, ZF, AF, PF, CF bayrakları sonuca göre yeniden düzenlenir.

<pre>if DF = 0 then
    SI = SI + 1
    DI = DI + 1
else
    SI = SI - 1
    DI = DI - 1</pre>
<table>
<tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr>
<tr><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td></tr>
</table>
</td>
</tr>
<tr>
<td>CMPSW</td>
<td>-</td>
<td>ES:[DI] ile DS:[SI] arasında bir word karşılatırır.

Algoritma:
<code>DS:[SI] - ES:[DI]</code>

OF, SF, ZF, AF, PF, CF bayrakları sonuca göre yeniden düzenlenir.

<pre>if DF = 0 then
    SI = SI + 2
    DI = DI + 2
else
    SI = SI - 2
    DI = DI - 2</pre>
<table>
<tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr>
<tr><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td></tr>
</table>
</td>
</tr>
</table>

CWD | - | İşaretli sayılarda word boyutunu doubleword boyutuna genişletir.

Algoritma:

```
if high bit of AX = 1 then
    DX = 65535 (0FFFFh)
else
    DX = 0
```

Örnek:

```
MOV DX, 0 ; DX = 0
MOV AX, 0 ; AX = 0
MOV AX, -5 ; DX AX = 00000h:0FFFBh
CWD ; DX AX = 0FFFFh:0FFFBh
RET
```

C	Z	S	O	P	A
değişmez					

DAA | - | Toplamadan sonra ondalık düzenlemesi yapar.
İki BCD değerinin toplam sonucunu düzenler.

Algoritma:

```
if low nibble of AL > 9 or AF = 1 then
    AL = AL + 6
    AF = 1
if AL > 9Fh or CF = 1 then
    AL = AL + 60h
    CF = 1
```

Örnek:

```
MOV AL, 0Fh       ; AL = 0Fh (15)
DAA               ; AL = 15h
RET
```

C	Z	S	O	P	A
r	r	r	r	r	r

DAS | - | Çıkarmadan sonra ondalık düzenlemesi yapar.
İki BCD değerinin çıkarma sonucunu düzenler.

Algoritma:

```
if low nibble of AL > 9 or AF = 1 then
    AL = AL - 6
    AF = 1
if AL > 9Fh or CF = 1 then
    AL = AL - 60h
    CF = 1
```

Örnek:

```
MOV AL, 0FFh      ; AL = 0FFh (-1)
DAS               ; AL = 99h, CF = 1
RET
```

C	Z	S	O	P	A
r	r	r	r	r	r

<table>
<tr>
<td>DEC</td>
<td>REG
memory</td>
<td>Azaltma işlemi yapar.

Algoritma:
<pre>operand = operand - 1</pre>
Örnek:
<pre>MOV AL, 255 ; AL = 0FFh (255 or -1)
DEC AL ; AL = 0FEh (254 or -2)
RET</pre>
<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td></tr></table>
CF değişmez!</td>
</tr>
<tr>
<td>DIV</td>
<td>REG
memory</td>
<td>İşaretsiz bölme yapar.

Algoritma:
<pre>operand byte ise:
    AL = AX / operand
    AH = remainder (kalan)
operand word ise:
    AX = (DX AX) / operand
    DX = remainder (kalan)</pre>
Örnek:
<pre>MOV AX, 203   ; AX = 00CBh
MOV BL, 4
DIV BL   ; AL = 50 (32h), AH = 3
RET</pre>
<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td>?</td><td>?</td><td>?</td><td>?</td><td>?</td><td>?</td></tr></table></td>
</tr>
<tr>
<td>HLT</td>
<td>-</td>
<td>Program kapatılır.

Örnek:
<pre>MOV AX, 5
HLT</pre>
<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td colspan="6">değişmez</td></tr></table></td>
</tr>
</table>

IDIV

REG
memory

İşaretli bölme yapar.

Algoritma:

```
operand byte ise:
    AL = AX / operand
    AH = remainder (kalan)
operand word ise:
    AX = (DX AX) / operand
    DX = remainder (kalan)
```

Örnek:

```
MOV AX, -203 ; AX = 0FF35h
MOV BL, 4
IDIV BL ; AL = -50 (0CEh), AH = -3 (0FDh)
RET
```

C	Z	S	O	P	A
?	?	?	?	?	?

IMUL

REG
memory

İşaretli çarpma yapar.

Algoritma:

```
operand byte ise:
    AX = AL * operand
operand word ise:
    (DX AX) = AX * operand
```

Örnek:

```
MOV AL, -2
MOV BL, -4
IMUL BL ; AX = 8
RET
```

C	Z	S	O	P	A
r	?	?	r	?	?

Sonuç operandan taşmazsa CF=OF=0 sıfır olur.

IN

AL, im.byte
AL, DX
AX, im.byte
AX, DX

AL veya AX'e porttan giriş alır.
İkinci operand port numarasıdır.

Second operand is a port number. Eğer 255 üstündeki portlara erişim gerekiyorsa DX register'ı kullanılır.

Örnek:

```
IN AX, 4 ; get status of traffic lights.
IN AL, 7 ; get status of stepper-motor.
```

C	Z	S	O	P	A
değişmez					

<table>
<tr>
<td>INC</td>
<td>REG
memory</td>
<td>Artırma işlemi yapar.

Algoritma:
<pre>operand = operand + 1</pre>Örnek:
<pre>MOV AL, 4
INC AL ; AL = 5
RET</pre>
<table>
<tr><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr>
<tr><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td></tr>
</table>

CF değişmez!</td>
</tr>
<tr>
<td>INT</td>
<td>immediate byte</td>
<td>Operandla verilen interrupt’ı çalıştırır.

Algoritma:
Stack’a push et:
bayrak register’ları
CS
IP
IF = 0
Program akışını interrupt prosedürüne atlatır

Örnek:
<pre>MOV AH, 0Eh ; teletype.
MOV AL, 'A'
INT 10h ; BIOS interrupt.
RET</pre>
<table>
<tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td><td>I</td></tr>
<tr><td colspan="6">değişmez</td><td>0</td></tr>
</table>
</td>
</tr>
<tr>
<td>INTO</td>
<td>-</td>
<td>Overflow bayrağı 1 ise interrupt 4’ü çalıştır.

Algoritma:
<pre>if OF = 1 then INT 4</pre>Örnek:
<pre>; -5 - 127 = -132 (not in -128..127)
; the result of SUB is wrong (124),
; so OF = 1 is set:
MOV AL, -5
SUB AL, 127 ; AL = 7Ch (124)
INTO ; process error.
RET</pre></td>
</tr>
</table>

<table>
<tr>
<td>IRET</td>
<td>-</td>
<td>Interrupt dönüşü.

Algoritma:

```
Stack'tan pop et:
    IP
    CS
    flags register
```


<table>
<tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr>
<tr><td colspan="6">pop edilir</td></tr>
</table>
</td>
</tr>
<tr>
<td>JA</td>
<td>etiket</td>
<td>Eğer birinci operand ikinciden büyükse kısa atlama (short jump yapar). İşaretsiz karşılaştırma yapar.

Algoritma:

```
if (CF = 0) and (ZF = 0) then jump
```


Örnek:

```
include 'emu8086.inc'
#make_COM#
ORG 100h
MOV AL, 250
CMP AL, 5
JA etiket1
PRINT 'AL 5'ten büyük değil'
JMP exit
etiket1:
    PRINT 'AL 5'ten büyük'
exit:
RET
```


<table>
<tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr>
<tr><td colspan="6">değişmez</td></tr>
</table>
</td>
</tr>
</table>

<table>
<tr>
<td>JAE</td>
<td>etiket</td>
<td>Eğer birinci operand ikinciden büyük veya eşitse kısa atlama (short jump yapar). İşaretsiz karşılaştırma yapar.

Algoritma:
<pre>if CF = 0 then jump</pre>
Örnek:
<pre>include 'emu8086.inc'
#make_COM#
ORG 100h
MOV AL, 5
CMP AL, 5
JAE etiket1
PRINT 'AL is not above or equal to 5'
JMP exit
etiket1:
    PRINT 'AL is above or equal to 5'
exit:
RET</pre>
<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td colspan="6">değişmez</td></tr></table></td>
</tr>
<tr>
<td>JB</td>
<td>etiket</td>
<td>Eğer birinci operand ikinciden küçükse kısa atlama (short jump yapar). İşaretsiz karşılaştırma yapar.

Algoritma:
<pre>if CF = 1 then jump</pre>
Örnek:
<pre>include 'emu8086.inc'
#make_COM#
ORG 100h
MOV AL, 1
CMP AL, 5
JB etiket1
PRINT 'AL is not below 5'
JMP exit
etiket1:
    PRINT 'AL is below 5'
exit:
RET</pre>
<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td colspan="6">değişmez</td></tr></table></td>
</tr>
</table>

<table>
<tr>
<td>JBE</td>
<td>etiket</td>
<td>Eğer birinci operand ikinciden küçük veya eşitse kısa atlama (short jump yapar). İşaretsiz karşılaştırma yapar.

Algoritma:
<pre>if CF = 1 or ZF = 1 then jump</pre>Örnek:
<pre>include 'emu8086.inc'
#make_COM#
ORG 100h
MOV AL, 5
CMP AL, 5
JBE etiket1
PRINT 'AL is not below or equal to 5'
JMP exit
etiket1:
    PRINT 'AL is below or equal to 5'
exit:
RET</pre><table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td colspan="6">değişmez</td></tr></table></td>
</tr>
<tr>
<td>JC</td>
<td>etiket</td>
<td>Taşma bayrak biti 1 ise kısa atlama yapar.

Algoritma:
<pre>if CF = 1 then jump</pre>Örnek:
<pre>include 'emu8086.inc'
#make_COM#
ORG 100h
MOV AL, 255
ADD AL, 1
JC etiket1
PRINT 'taşma yok'
JMP exit
etiket1:
    PRINT 'taşma var'
exit:
RET</pre><table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td colspan="6">değişmez</td></tr></table></td>
</tr>
</table>

<table>
<tr>
<td>JCXZ</td>
<td>etiket</td>
<td>CX register’ı 1 ise kısa atlama yapar.

Algoritma:
<pre>if CX = 0 then jump</pre>
Örnek:
<pre>include 'emu8086.inc'
#make_COM#
ORG 100h
MOV CX, 0
JCXZ etiket1
PRINT 'CX sıfır değildir.'
JMP exit
etiket1:
    PRINT 'CX sıfırdır.'
exit:
RET</pre>
<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td colspan="6">değişmez</td></tr></table></td>
</tr>
<tr>
<td>JE</td>
<td>etiket</td>
<td>Eğer birinci operand ikinci operanda eşitse kısa atlama (short jump yapar). İşaretli veya işaretsiz karşılaştırma yapar.

Algoritma:
<pre>if ZF = 1 then jump</pre>
Örnek:
<pre>include 'emu8086.inc'
#make_COM#
ORG 100h
MOV AL, 5
CMP AL, 5
JE etiket1
PRINT 'AL 5’e eşit değil.'
JMP exit
etiket1:
    PRINT 'AL 5’e eşit.'
exit:
RET</pre>
<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td colspan="6">değişmez</td></tr></table></td>
</tr>
</table>

<table>
<tr>
<td>JG</td>
<td>etiket</td>
<td>Eğer birinci operand ikinciden büyük veya eşitse kısa atlama (short jump yapar). İşaretli karşılaştırma yapar.

Algoritma:
<pre>if (ZF = 0) and (SF = OF) then jump</pre>Örnek:
<pre>include 'emu8086.inc'
#make_COM#
ORG 100h
MOV AL, 5
CMP AL, -5
JG etiket1
PRINT 'AL -5'ten büyük değildir.'
JMP exit
etiket1:
    PRINT ' AL -5'ten büyüktür.'
exit:
RET</pre><table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td colspan="6">değişmez</td></tr></table></td>
</tr>
<tr>
<td>JGE</td>
<td>etiket</td>
<td>Eğer birinci operand ikinciden büyük veya eşitse kısa atlama (short jump yapar). İşaretli karşılaştırma yapar.

Algoritma:
<pre>if SF = OF then jump</pre>Örnek:
<pre>include 'emu8086.inc'
#make_COM#
ORG 100h
MOV AL, 2
CMP AL, -5
JGE etiket1
PRINT 'AL < -5'
JMP exit
etiket1:
    PRINT 'AL >= -5'
exit:
RET</pre><table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td colspan="6">değişmez</td></tr></table></td>
</tr>
</table>

<table>
<tr>
<td>JL</td>
<td>etiket</td>
<td>Eğer birinci operand ikinciden küçükse kısa atlama (short jump yapar). İşaretli karşılaştırma yapar.

Algoritma:
<code>if SF <> OF then jump</code>

Örnek:
<pre>include 'emu8086.inc'
#make_COM#
ORG 100h
MOV AL, -2
CMP AL, 5
JL etiket1
PRINT 'AL >= 5.'
JMP exit
etiket1:
    PRINT 'AL < 5.'
exit:
RET</pre>
<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td colspan="6">değişmez</td></tr></table></td>
</tr>
<tr>
<td>JLE</td>
<td>etiket</td>
<td>Eğer birinci operand ikinciden küçük veya eşitse kısa atlama (short jump yapar). İşaretli karşılaştırma yapar.

Algoritma:
<code>if SF <> OF or ZF = 1 then jump</code>

Örnek:
<pre>include 'emu8086.inc'
#make_COM#
ORG 100h
MOV AL, -2
CMP AL, 5
JLE etiket1
PRINT 'AL > 5.'
JMP exit
etiket1:
    PRINT 'AL <= 5.'
exit:
RET</pre>
<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td colspan="6">değişmez</td></tr></table></td>
</tr>
</table>

<table>
<tr>
<td>JMP</td>
<td>etiket
4-byte adres</td>
<td>Şartsız atlama yapar.
4-byte adres 1234h:5678h şeklinde girilebilir. İlk değer segment ikinci offset adrestir.

Algoritma:
<pre>always jump</pre>
Örnek:
<pre>include 'emu8086.inc'
#make_COM#
ORG 100h
MOV AL, 5
JMP etiket1 ; jump over 2 lines!
PRINT 'Not Jumped!'
MOV AL, 0
etiket1:
PRINT 'Got Here!'
RET</pre>
<table>
<tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr>
<tr><td colspan="6">değişmez</td></tr>
</table>
</td>
</tr>
<tr>
<td>JNA</td>
<td>etiket</td>
<td>Eğer birinci operand ikinciden büyük değilse kısa atlama (short jump) yapar. İşaretsiz karşılaştırma yapar.

Algoritma:
<pre>if CF = 1 or ZF = 1 then jump</pre>
Örnek:
<pre>include 'emu8086.inc'
#make_COM#
ORG 100h
MOV AL, 2
CMP AL, 5
JNA etiket1
PRINT 'AL is above 5.'
JMP exit
etiket1:
PRINT 'AL is not above 5.'
exit:
RET</pre>
<table>
<tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr>
<tr><td colspan="6">değişmez</td></tr>
</table>
</td>
</tr>
</table>

<table>
<tr>
<td>JNAE</td>
<td>etiket</td>
<td>Eğer birinci operand ikinciden büyük değilse ve eşit değilse kısa atlama (short jump) yapar. İşaretsiz karşılaştırma yapar.

Algoritma:
<pre>if CF = 1 then jump</pre>
Örnek:
<pre>include 'emu8086.inc'
#make_COM#
ORG 100h
MOV AL, 2
CMP AL, 5
JNAE etiket1
PRINT 'AL >= 5.'
JMP exit
etiket1:
    PRINT 'AL < 5.'
exit:
RET</pre>
<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td colspan="6">değişmez</td></tr></table></td>
</tr>
<tr>
<td>JNB</td>
<td>etiket</td>
<td>Eğer birinci operand ikinciden küçük değilse kısa atlama (short jump) yapar. İşaretsiz karşılaştırma yapar.

Algoritma:
<pre>if CF = 0 then jump</pre>
Örnek:
<pre>include 'emu8086.inc'
#make_COM#
ORG 100h
MOV AL, 7
CMP AL, 5
JNB etiket1
PRINT 'AL < 5.'
JMP exit
etiket1:
PRINT 'AL >= 5.'
exit:
RET</pre>
<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td colspan="6">değişmez</td></tr></table></td>
</tr>
</table>

JNBE — etiket

Eğer birinci operand ikinciden küçük değilse ve eşit değilse kısa atlama (short jump) yapar. İşaretsiz karşılaştırma yapar.

Algoritma:

```
if (CF = 0) and (ZF = 0) then jump
```

Örnek:

```
include 'emu8086.inc'
#make_COM#
ORG 100h
MOV AL, 7
CMP AL, 5
JNBE etiket1
PRINT 'AL <= 5.'
JMP exit
etiket1:
    PRINT 'AL > 5.'
exit:
RET
```

C	Z	S	O	P	A
değişmez					

JNC — etiket

Taşma bayrak biti (carry flag) 0 ise kısa atlama (short jump) yapar.

Algoritma:

```
if CF = 0 then jump
```

Örnek:

```
include 'emu8086.inc'
#make_COM#
ORG 100h
MOV AL, 2
ADD AL, 3
JNC etiket1
PRINT 'has carry.'
JMP exit
etiket1:
    PRINT 'no carry.'
exit:
RET
```

C	Z	S	O	P	A
değişmez					

<table>
<tr>
<td>JNE</td>
<td>etiket</td>
<td>Eğer birinci operand ikinciye eşit değilse kısa atlama (short jump) yapar. İşaretli veya işaretsiz karşılaştırma yapar.

Algoritma:
<pre>if ZF = 0 then jump</pre>
Örnek:
<pre>include 'emu8086.inc'
#make_COM#
ORG 100h
MOV AL, 2
CMP AL, 3
JNE etiket1
PRINT 'AL = 3.'
JMP exit
etiket1:
    PRINT 'Al <> 3.'
exit:
RET</pre>
<table>
<tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr>
<tr><td colspan="6">değişmez</td></tr>
</table>
</td>
</tr>
<tr>
<td>JNG</td>
<td>etiket</td>
<td>Eğer birinci operand ikinciden büyük değilse kısa atlama (short jump) yapar. İşaretli karşılaştırma yapar.

Algoritma:
<pre>if (ZF = 1) and (SF <> OF) then jump</pre>
Örnek:
<pre>include 'emu8086.inc'
#make_COM#
ORG 100h
MOV AL, 2
CMP AL, 3
JNG etiket1
PRINT 'AL > 3.'
JMP exit
etiket1:
PRINT 'Al <= 3.'
exit:
RET</pre>
<table>
<tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr>
<tr><td colspan="6">değişmez</td></tr>
</table>
</td>
</tr>
</table>

<table>
<tr>
<td>JNGE</td>
<td>etiket</td>
<td>Eğer birinci operand ikinciden büyük değilse ve eşit değilse kısa atlama (short jump) yapar. İşaretli karşılaştırma yapar.

Algoritma:
<pre>if SF <> OF then jump</pre>
Örnek:
<pre>include 'emu8086.inc'
#make_COM#
ORG 100h
MOV AL, 2
CMP AL, 3
JNGE etiket1
PRINT 'AL >= 3.'
JMP exit
etiket1:
    PRINT 'Al < 3.'
exit:
RET</pre>
<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td colspan="6">değişmez</td></tr></table></td>
</tr>
<tr>
<td>JNL</td>
<td>etiket</td>
<td>Eğer birinci operand ikinciden küçük değilse kısa atlama (short jump) yapar. İşaretli karşılaştırma yapar.

Algoritma:
<pre>if SF = OF then jump</pre>
Örnek:
<pre>include 'emu8086.inc'
#make_COM#
ORG 100h
MOV AL, 2
CMP AL, -3
JNL etiket1
PRINT 'AL < -3.'
JMP exit
etiket1:
PRINT 'Al >= -3.'
exit:
RET</pre>
<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td colspan="6">değişmez</td></tr></table></td>
</tr>
</table>

<table>
<tr>
<td>JNLE</td>
<td>etiket</td>
<td>Eğer birinci operand ikinciden küçük değilse ve eşit değilse kısa atlama (short jump) yapar. İşaretli karşılaştırma yapar.

Algoritma:
<pre>if (SF = OF) and (ZF = 0) then jump</pre>
Örnek:
<pre>include 'emu8086.inc'
#make_COM#
ORG 100h
MOV AL, 2
CMP AL, -3
JNLE etiket1
PRINT 'AL <= -3.'
JMP exit
etiket1:
    PRINT 'Al > -3.'
exit:
RET</pre>
<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td colspan="6">değişmez</td></tr></table></td>
</tr>
<tr>
<td>JNO</td>
<td>etiket</td>
<td>Overflow bayrak biti 0 ise kısa atlama (short jump) yapar.

Algoritma:
<pre>if OF = 0 then jump</pre>
Örnek:
<pre>; -5 - 2 = -7 (inside -128..127)
; the result of SUB is correct,
; so OF = 0:
include 'emu8086.inc'
#make_COM#
ORG 100h
MOV AL, -5
SUB AL, 2 ; AL = 0F9h (-7)
JNO etiket1
PRINT 'overflow!'
JMP exit
etiket1:
    PRINT 'no overflow.'
exit:
RET</pre>
<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td colspan="6">değişmez</td></tr></table></td>
</tr>
</table>

<table>
<tr>
<td>JNP</td>
<td>etiket</td>
<td>Parity (eşlik) bayrak biti 0 ise (odd-tek) kısa atlama (short jump) yapar. Sadece en düşük öneme sahip 8 bit kontrol edilir. CMP, SUB, ADD, TEST, AND, OR, XOR komutları set eder.

Algoritma:
<pre>if PF = 0 then jump</pre>Örnek:
<pre>include 'emu8086.inc'
#make_COM#
ORG 100h
MOV AL, 00000111b ; AL = 7
OR AL, 0 ; just set flags.
JNP etiket1
PRINT 'parity even.'
JMP exit
etiket1:
PRINT 'parity odd.'
exit:
RET</pre><table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td colspan="6">değişmez</td></tr></table></td>
</tr>
<tr>
<td>JNS</td>
<td>etiket</td>
<td>İşaret bayrak (signed) biti 1 değilse (pozitif) kısa atlama (short jump) yapar. CMP, SUB, ADD, TEST, AND, OR, XOR komutları set eder.

Algoritma:
<pre>if SF = 0 then jump</pre>Örnek:
<pre>include 'emu8086.inc'
#make_COM#
ORG 100h
MOV AL, 00000111b ; AL = 7
OR AL, 0 ; just set flags.
JNS etiket1
PRINT 'signed.'
JMP exit
etiket1:
    PRINT 'not signed.'
exit:
RET</pre><table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td colspan="6">değişmez</td></tr></table></td>
</tr>
</table>

JNZ | etiket

Sıfır bayrak (ZF) biti 0 ise (pozitif) kısa atlama (short jump) yapar.
CMP, SUB, ADD, TEST, AND, OR, XOR komutları set eder.

Algoritma:
```
if ZF = 0 then jump
```

Örnek:
```
include 'emu8086.inc'
#make_COM#
ORG 100h
MOV AL, 00000111b ; AL = 7
OR AL, 0 ; just set flags.
JNZ etiket1
PRINT 'zero.'
JMP exit
etiket1:
PRINT 'not zero.'
exit:
RET
```

C	Z	S	O	P	A
değişmez					

JO | etiket

Overflow bayrak (OF) biti 1 ise kısa atlama (short jump) yapar.

Algoritma:
```
if OF = 1 then jump
```

Örnek:
```
; -5 - 127 = -132 (not in -128..127)
; the result of SUB is wrong (124),
; so OF = 1 is set:
include 'emu8086.inc'
#make_COM#
org 100h
MOV AL, -5
SUB AL, 127 ; AL = 7Ch (124)
JO etiket1
PRINT 'no overflow.'
JMP exit
etiket1:
PRINT 'overflow!'
exit:
RET
```

C	Z	S	O	P	A
değişmez					

<table>
<tr>
<td>JP</td>
<td>etiket</td>
<td>Parity (eşlik) bayrak biti 1 ise (even-çift) kısa atlama (short jump) yapar. Sadece en düşük öneme sahip 8 bit kontrol edilir. CMP, SUB, ADD, TEST, AND, OR, XOR komutları set eder.

Algoritma:
<pre>if PF = 1 then jump</pre>Örnek:
<pre>include 'emu8086.inc'
#make_COM#
ORG 100h
MOV AL, 00000101b ; AL = 5
OR AL, 0 ; just set flags.
JP etiket1
PRINT 'parity odd.'
JMP exit
etiket1:
PRINT 'parity even.'
exit:
RET</pre><table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td colspan="6">değişmez</td></tr></table></td>
</tr>
<tr>
<td>JPE</td>
<td>etiket</td>
<td>Parity çift ise (even) kısa atlama (short jump) yapar. Sadece en düşük öneme sahip 8 bit kontrol edilir. CMP, SUB, ADD, TEST, AND, OR, XOR komutları set eder.

Algoritma:
<pre>if PF = 1 then jump</pre>Örnek:
<pre>include 'emu8086.inc'
#make_COM#
ORG 100h
MOV AL, 00000101b ; AL = 5
OR AL, 0 ; just set flags.
JPE etiket1
PRINT 'parity odd.'
JMP exit
etiket1:
PRINT 'parity even.'
exit:
RET</pre><table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td colspan="6">değişmez</td></tr></table></td>
</tr>
</table>

<table>
<tr>
<td>JPO</td>
<td>etiket</td>
<td>Parity tek ise (odd) kısa atlama (short jump) yapar. Sadece en düşük öneme sahip 8 bit kontrol edilir. CMP, SUB, ADD, TEST, AND, OR, XOR komutları set eder.

Algoritma:

```
if PF = 0 then jump
```


Örnek:

```
include 'emu8086.inc'
#make_COM#
ORG 100h
MOV AL, 00000111b ; AL = 7
OR AL, 0 ; just set flags.
JPO etiket1
PRINT 'parity even.'
JMP exit
etiket1:
PRINT 'parity odd.'
exit:
RET
```


<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td colspan="6">değişmez</td></tr></table></td>
</tr>
<tr>
<td>JS</td>
<td>etiket</td>
<td>Sign bayrak biti 1 ise (negatif) kısa atlama (short jump) yapar. CMP, SUB, ADD, TEST, AND, OR, XOR komutları set eder.

Algoritma:

```
if SF = 1 then jump
```


Örnek:

```
include 'emu8086.inc'
#make_COM#
ORG 100h
MOV AL, 10000000b ; AL = -128
OR AL, 0 ; just set flags.
JS etiket1
PRINT 'not signed.'
JMP exit
etiket1:
    PRINT 'signed.'
exit:
RET
```


<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td colspan="6">değişmez</td></tr></table></td>
</tr>
</table>

<table>
<tr>
<td>JZ</td>
<td>etiket</td>
<td>Eğer sıfır ise (ZF=1) kısa atlama (short jump) yapar. CMP, SUB, ADD, TEST, AND, OR, XOR komutları set eder.

Algoritma:
<pre>if ZF = 1 then jump</pre>
Örnek:
<pre>include 'emu8086.inc'
#make_COM#
ORG 100h
MOV AL, 5
CMP AL, 5
JZ etiket1
PRINT 'AL is not equal to 5.'
JMP exit
etiket1:
PRINT 'AL is equal to 5.'
exit:
RET</pre>
<table>
<tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr>
<tr><td colspan="6">değişmez</td></tr>
</table>
</td>
</tr>
<tr>
<td>LAHF</td>
<td>-</td>
<td>AH register’ına bayrak register’ının (flags register) en düşük öneme sahip 8 bit’ini yükler.

Algoritma:
<pre>AH = flags register
AH: 7   6   5   4   3   2   1   0
      [SF] [ZF] [0] [AF] [0] [PF] [1] [CF]
1, 3, 5 bitleri ayrılmıştır.</pre>
<table>
<tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr>
<tr><td colspan="6">değişmez</td></tr>
</table>
</td>
</tr>
<tr>
<td>LDS</td>
<td>REG, memory</td>
<td>Hafızadan alınan doubleword datayı hedef operand register’a ve DS’ye yükler.

Algoritma:
<pre>REG = first word
DS = second word</pre>
Örnek:
<pre>#make_COM#
ORG 100h
LDS AX, m
RET
m DW 1234h
DW 5678h
END
AX is set to 1234h, DS is set to 5678h.</pre>
<table>
<tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr>
<tr><td colspan="6">değişmez</td></tr>
</table>
</td>
</tr>
</table>

<table>
<tr>
<td>LEA</td>
<td>REG, memory</td>
<td>Efektif adres yükler.

Algoritma:
<pre>REG = adres of memory (offset)</pre>
Örnek:
<pre>#make_COM#
ORG 100h
LEA AX, m
RET
m DW 1234h
END</pre>
AX’e 0104h değeri atanır.
LEA komutu 3 byte, RET komutu 1 byte’tır. 0100h adresinden başlandığı için m’nin adresi 0104h olur.

<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td colspan="6">değişmez</td></tr></table></td>
</tr>
<tr>
<td>LES</td>
<td>REG, memory</td>
<td>Hafızadan alınan doubleword datayı hedef operand register’a ve ES’ye yükler.

Algoritma:
<pre>REG = first word
ES = second word</pre>
Örnek:
<pre>#make_COM#
ORG 100h
LES AX, m
RET
m DW 1234h
DW 5678h
END</pre>
AX register’ının değeri 1234h olur ve ES register’ının değeri 5678h olur.

<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td colspan="6">değişmez</td></tr></table></td>
</tr>
</table>

LODSB | - | AL register'ına DS:[SI] adresinden bir byte yükler. SI değeri güncellenir (1 artırılır (D=0), 1 azaltılır (D=1))

Algoritma:

```
AL = DS:[SI]
if DF = 0 then
SI = SI + 1
else
SI = SI - 1
```

Örnek:

```
#make_COM#
ORG 100h
LEA SI, a1
MOV CX, 5
MOV AH, 0Eh
m: LODSB
INT 10h
LOOP m
RET
a1 DB 'H', 'e', 'l', 'l', 'o'
```

C	Z	S	O	P	A
değişmez					

LODSW | - | AX register'ına DS:[SI] adresinden bir word yükler. SI değeri güncellenir (2 artırılır (D=0), 2 azaltılır (D=1))

Algoritma:

```
AX = DS:[SI]
if DF = 0 then
SI = SI + 2
else
SI = SI - 2
```

Örnek:

```
#make_COM#
ORG 100h
LEA SI, a1
MOV CX, 5
REP LODSW ; finally there will be 555h in AX.
RET
a1 dw 111h, 222h, 333h, 444h, 555h
```

C	Z	S	O	P	A
değişmez					

<table>
<tr>
<td>LOOP</td>
<td>etiket</td>
<td>CX bir azaltılır ve CX sıfır değilse etikete atlanır.

Algoritma:
<pre>CX = CX - 1
if CX <> 0 then
jump
else
no jump, continue</pre>
Örnek:
<pre>include 'emu8086.inc'
#make_COM#
ORG 100h
MOV CX, 5
etiket1:
PRINTN 'loop!'
LOOP etiket1
RET</pre>
<table>
<tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr>
<tr><td colspan="6">değişmez</td></tr>
</table>
</td>
</tr>
<tr>
<td>LOOPE</td>
<td>etiket</td>
<td>CX bir azaltılır ve CX sıfır değilse ve eşitlik varsa (verilen iki operand arasında) (ZF=1) etikete atlar.

Algoritma:
<pre>CX = CX - 1
if (CX <> 0) and (ZF = 1) then
jump
else
no jump, continue</pre>
Örnek:
<pre>; Loop until result fits into AL alone,
; or 5 times. The result will be over 255
; on third loop (100+100+100),
; so loop will exit.
include 'emu8086.inc'
#make_COM#
ORG 100h
MOV AX, 0
MOV CX, 5
etiket1:
PUTC '*'
ADD AX, 100
CMP AH, 0
LOOPE etiket1
RET</pre>
<table>
<tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr>
<tr><td colspan="6">değişmez</td></tr>
</table>
</td>
</tr>
</table>

LOOPNE — etiket

CX bir azaltılır ve CX sıfır değilse ve eşitlik yoksa (verilen iki operand arasında) (ZF=0)

Algoritma:

```
CX = CX - 1
if (CX <> 0) and (ZF = 0) then
jump
else
no jump, continue
```

Örnek:

```
; Loop until '7' is found,
; or 5 times.
include 'emu8086.inc'
#make_COM#
ORG 100h
MOV SI, 0
MOV CX, 5
etiket1:
PUTC '*'
MOV AL, v1[SI]
INC SI ; next byte (SI=SI+1).
CMP AL, 7
LOOPNE etiket1
RET
v1 db 9, 8, 7, 6, 5
```

C	Z	S	O	P	A
değişmez					

LOOPNZ — etiket

CX bir azaltılır ve CX sıfır değilse ve ZF=0 ise etikete atlar.

Algoritma:

```
CX = CX - 1
if (CX <> 0) and (ZF = 0) then
jump
else
no jump, continue
```

Örnek:

```
; Loop until '7' is found,
; or 5 times.
include 'emu8086.inc'
#make_COM#
ORG 100h
MOV SI, 0
MOV CX, 5
etiket1:
PUTC '*'
MOV AL, v1[SI]
INC SI ; next byte (SI=SI+1).
CMP AL, 7
LOOPNZ etiket1
RET
v1 db 9, 8, 7, 6, 5
```

C	Z	S	O	P	A
değişmez					

<table>
<tr>
<td>LOOPZ</td>
<td>etiket</td>
<td>CX bir azaltılır ve CX sıfır değilse ve ZF=1 ise etikete atlar.

Algoritma:

```
CX = CX - 1
if (CX <> 0) and (ZF = 1) then
jump
else
no jump, continue
```


Örnek:

```
; Loop until result fits into AL alone,
; or 5 times. The result will be over 255
; on third loop (100+100+100),
; so loop will exit.
include 'emu8086.inc'
#make_COM#
ORG 100h
MOV AX, 0
MOV CX, 5
etiket1:
PUTC '*'
ADD AX, 100
CMP AH, 0
LOOPZ etiket1
RET
```


<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td colspan="6">değişmez</td></tr></table></td>
</tr>
<tr>
<td>MOV</td>
<td>REG, memory
memory, REG
REG, REG
memory, immediate
REG, immediate
SREG, memory
memory, SREG
REG, SREG
SREG, REG</td>
<td>İkinci operand birinci operanda kopyalanır.

MOV komutuyla:

CS ve IP register’larına değer atanamaz.
İki segment register’ı arasında değer aktarılamaz (önce genel amaçlı register’a aktarılmalıdır).

Algoritma:

```
operand1 = operand2
```


Örnek:

```
#make_COM#
ORG 100h
MOV AX, 0B800h ; set AX = B800h (VGA memory).
MOV DS, AX ; copy value of AX to DS.
MOV CL, 'A' ; CL = 41h (ASCII code).
MOV CH, 01011111b ; CL = color attribute.
MOV BX, 15Eh ; BX = position on screen.
MOV [BX], CX ; w.[0B800h:015Eh] = CX.
RET ; returns to operating system.
```


<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td colspan="6">değişmez</td></tr></table></td>
</tr>
</table>

MOVSB | - | DS:[SI] adresinden ES:[DI] adresine bir byte kopyalar. SI ve DI register'ları güncellenir.

Algoritma:

```
ES:[DI] = DS:[SI]
if DF = 0 then
    SI = SI + 1
    DI = DI + 1
else
    SI = SI - 1
    DI = DI - 1
```

Örnek:

```
#make_COM#
ORG 100h
LEA SI, a1
LEA DI, a2
MOV CX, 5
REP MOVSB
RET
a1 DB 1,2,3,4,5
a2 DB 5 DUP(0)
```

C	Z	S	O	P	A
değişmez					

MOVSW | - | DS:[SI] adresinden ES:[DI] adresine bir word kopyalar. SI ve DI register'ları güncellenir.

Algoritma:

```
ES:[DI] = DS:[SI]
if DF = 0 then
    SI = SI + 2
    DI = DI + 2
else
    SI = SI - 2
    DI = DI - 2
```

Örnek:

```
#make_COM#
ORG 100h
LEA SI, a1
LEA DI, a2
MOV CX, 5
REP MOVSW
RET
a1 DW 1,2,3,4,5
a2 DW 5 DUP(0)
```

C	Z	S	O	P	A
değişmez					

<table>
<tr>
<td>MUL</td>
<td>REG
memory</td>
<td>İşaretsiz çarpma yapar.

Algoritma:
<pre>operand byte ise:
AX = AL * operand.
operand word ise:
(DX AX) = AX * operand.</pre>
Örnek:
<pre>MOV AL, 200 ; AL = 0C8h
MOV BL, 4
MUL BL ; AX = 0320h (800)
RET</pre>
<table>
<tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr>
<tr><td>r</td><td>?</td><td>?</td><td>r</td><td>?</td><td>?</td></tr>
</table>
CF=OF=0 when high section of the result is zero.</td>
</tr>
<tr>
<td>NEG</td>
<td>REG
memory</td>
<td>Bir sayının işaretini tersler (ikinin tümleyeni).

Algoritma:
<pre>Invert all bits of the operand
Add 1 to inverted operand</pre>
Örnek:
<pre>MOV AL, 5 ; AL = 05h
NEG AL ; AL = 0FBh (-5)
NEG AL ; AL = 05h (5)
RET</pre>
<table>
<tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr>
<tr><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td></tr>
</table></td>
</tr>
<tr>
<td>NOP</td>
<td>-</td>
<td>İşlem yapmaz.

Algoritma:
<pre>Do nothing</pre>
Örnek:
<pre>; do nothing, 3 times:
NOP
NOP
NOP
RET</pre>
<table>
<tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr>
<tr><td colspan="6">değişmez</td></tr>
</table></td>
</tr>
</table>

<table>
<tr>
<td>NOT</td>
<td>REG
memory</td>
<td>Operandaki her bit terslenir (birin tümleyeni).

Algoritma:

```
if bit is 1 turn it to 0.
if bit is 0 turn it to 1.
```


Örnek:

```
MOV AL, 00011011b
NOT AL ; AL = 11100100b
RET
```


<table>
<tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr>
<tr><td colspan="6">değişmez</td></tr>
</table>
</td>
</tr>
<tr>
<td>OR</td>
<td>REG, memory
memory, REG
REG, REG
memory,
immediate
REG, immediate</td>
<td>İki operandaki karşılıklı her bit için mantıksal veya işlemi yapar.

Doğruluk tablosu:

```
1 OR 1 = 1
1 OR 0 = 1
0 OR 1 = 1
0 OR 0 = 0
```


Örnek:

```
MOV AL, 'A' ; AL = 01000001b
OR AL, 00100000b ; AL = 01100001b ('a')
RET
```


<table>
<tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr>
<tr><td>0</td><td>r</td><td>r</td><td>0</td><td>r</td><td>?</td></tr>
</table>
</td>
</tr>
<tr>
<td>OUT</td>
<td>im.byte, AL
im.byte, AX
DX, AL
DX, AX</td>
<td>AL veya AX register’ı verilen porta gönderilir.
İlk operand port numarasıdır. Eğer 255’in üstündeki portlara erişim gerekirse DX register’ı kullanılır.

Örnek:

```
MOV AX, 0FFFh ; Turn on all
OUT 4, AX ; traffic lights.
MOV AL, 100b ; Turn on the third
OUT 7, AL ; magnet of the stepper-motor.
```


<table>
<tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr>
<tr><td colspan="6">değişmez</td></tr>
</table>
</td>
</tr>
</table>

<table>
<tr>
<td>POP</td>
<td>REG
SREG
memory</td>
<td>Stack’tan 16 bit değer alır ve operanda aktarır.

Algoritma:
<pre>operand = SS:[SP] (stack’ın en üstü)
SP = SP + 2</pre>Örnek:
<pre>MOV AX, 1234h
PUSH AX
POP DX ; DX = 1234h
RET</pre><table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td colspan="6">değişmez</td></tr></table></td>
</tr>
<tr>
<td>POPA</td>
<td>-</td>
<td>Bütün genel amaçlı register’ları (DI, SI, BP, SP, BX, DX, CX, AX) stack’tan pop eder. SP değeri gözardı edilir.

Not: Bu komut sadece 80186 ve üstü lişlemcilerde kullanılır.

Algoritma:
<pre>POP DI
POP SI
POP BP
POP xx (SP gözardı edilir)
POP BX
POP DX
POP CX
POP AX</pre><table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td colspan="6">değişmez</td></tr></table></td>
</tr>
<tr>
<td>POPF</td>
<td>-</td>
<td>Flag register’ına stack’tan değer pop edilir.

Algoritma:
<pre>flags = SS:[SP] (top of the stack)
SP = SP + 2</pre><table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td colspan="6">Pop edilir</td></tr></table></td>
</tr>
</table>

<table>
<tr>
<td>PUSH</td>
<td>REG
SREG
memory
immediate</td>
<td>16 bit değer stack’a saklanır.
Store 16 bit value in the stack.

Not: PUSH immediate sadece 80186 ve üstü işlemcilerde kullanılır.

Algoritma:
<pre>SP = SP - 2
SS:[SP] (top of the stack) = operand</pre>
Örnek:
<pre>MOV AX, 1234h
PUSH AX
POP DX ; DX = 1234h
RET</pre>
<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td colspan="6">değişmez</td></tr></table></td>
</tr>
<tr>
<td>PUSHA</td>
<td>-</td>
<td>Tüm genel amaçlı register’ları (AX, CX, DX, BX, SP,
BP, SI, DI) stack’a push eder. SP register’ını PUSHA komutundan önceki değeri kullanılır.

Not: Bu komut sadece 80186 ve üstü işlemcilerde kullanılır.

Algoritma:
<pre>PUSH AX
PUSH CX
PUSH DX
PUSH BX
PUSH SP
PUSH BP
PUSH SI
PUSH DI</pre>
<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td colspan="6">değişmez</td></tr></table></td>
</tr>
<tr>
<td>PUSHF</td>
<td>-</td>
<td>Flag register’ı stack’a saklar.

Algoritma:
<pre>SP = SP - 2
SS:[SP] (top of the stack) = flags</pre>
<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td colspan="6">değişmez</td></tr></table></td>
</tr>
</table>

<table>
<tr>
<td>RCL</td>
<td>memory, immediate
REG, immediate
memory, CL
REG, CL</td>
<td>Birinci operandı taşma bayrağı (carry flag) üzerinden sola döndürür. Bit olarak dönme sayısı ikinci operandla belirtilir.

Algoritma:
<pre>shift all bits left, the bit that goes off is set to CF and
previous value of CF is inserted to the right-most position.</pre>
Örnek:
<pre>STC ; set carry (CF=1).
MOV AL, 1Ch ; AL = 00011100b
RCL AL, 1 ; AL = 00111001b, CF=0.
RET</pre>
<table>
<tr><td>C</td><td>O</td></tr>
<tr><td>r</td><td>r</td></tr>
</table>
OF=0 olur eğer birinci operand işaretini korursa.</td>
</tr>
<tr>
<td>RCR</td>
<td>memory, immediate
REG, immediate
memory, CL
REG, CL</td>
<td>Birinci operandı taşma bayrağı (carry flag) üzerinden sağa döndürür. Bit olarak dönme sayısı ikinci operandla belirtilir.

Algoritma:
<pre>shift all bits right, the bit that goes off is set to CF
and previous value of CF is inserted to the left-most position.</pre>
Örnek:
<pre>STC ; set carry (CF=1).
MOV AL, 1Ch ; AL = 00011100b
RCR AL, 1 ; AL = 10001110b, CF=0.
RET</pre>
<table>
<tr><td>C</td><td>O</td></tr>
<tr><td>r</td><td>r</td></tr>
</table>
OF=0 olur eğer birinci operand işaretini korursa.</td>
</tr>
<tr>
<td>REP</td>
<td>tekrarlanacak komut</td>
<td>MOVSB, MOVSW, LODSB, LODSW, STOSB, STOSW komutlarını CX defa tekrarlar.

Algoritma:
<pre>check_cx:
if CX <> 0 then
do following chain instruction
CX = CX - 1
go back to check_cx
else
exit from REP cycle</pre>
<table>
<tr><td>Z</td></tr>
<tr><td>r</td></tr>
</table></td>
</tr>
</table>

REPE	tekrarlanacak komut	CMPSB, CMPSW, SCASB, SCASW komutlarını ZF=1 olduğu sürece (sonuç eşit olur) ve en fazla CX değeri kadar tekrarlar.
		
		Algoritma:
		`check_cx:`
		`if CX <> 0 then`
		`do following chain instruction`
		`CX = CX - 1`
		`if ZF = 1 then`
		`m go back to check_cx`
		`else`
		`m exit from REPE cycle`
		`else`
		`exit from REPE cycle`
		
		Z
		r

REPNE	tekrarlanacak komut	CMPSB, CMPSW, SCASB, SCASW komutlarını ZF=0 olduğu sürece (sonuç eşit değil) ve en fazla CX değeri kadar tekrarlar.
		
		Algoritma:
		`check_cx:`
		`if CX <> 0 then`
		`do following chain instruction`
		`CX = CX - 1`
		`if ZF = 0 then`
		`go back to check_cx`
		`else`
		`m exit from REPNE cycle`
		`else`
		`exit from REPNE cycle`
		
		Z
		r

REPNZ	tekrarlanacak komut	CMPSB, CMPSW, SCASB, SCASW komutlarını ZF=0 olduğu sürece (sonuç sıfır değil) ve en fazla CX değeri kadar tekrarlar.
		
		Algoritma:
		`check_cx:`
		`if CX <> 0 then`
		`do following chain instruction`
		`CX = CX - 1`
		`if ZF = 0 then`
		`m go back to check_cx`
		`else`
		`m exit from REPNZ cycle`
		`else`
		`exit from REPNZ cycle`
		
		Z
		r

<table>
<tr>
<td>REPZ</td>
<td>tekrarlanacak komut</td>
<td>CMPSB, CMPSW, SCASB, SCASW komutlarını ZF=1 olduğu sürece (sonuç sıfır) ve en fazla CX değeri kadar tekrarlar.

Algoritma:
<pre>check_cx:
if CX <> 0 then
do following chain instruction
CX = CX - 1
if ZF = 1 then
go back to check_cx
else
exit from REPZ cycle
else
exit from REPZ cycle</pre>
<table><tr><td>Z</td></tr><tr><td>r</td></tr></table></td>
</tr>
<tr>
<td>RET</td>
<td>-</td>
<td>Near prosedürden dönüş yapar.

Algoritma:
<pre>Pop from stack:
IP
if immediate operand is present: SP = SP + operand</pre>
Örnek:
<pre>#make_COM#
ORG 100h ; for COM file.
CALL p1
ADD AX, 1
RET ; return to OS.

p1 PROC ; procedure declaration.
MOV AX, 1234h
RET ; return to caller.
p1 ENDP</pre>
<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td colspan="6">değişmez</td></tr></table></td>
</tr>
<tr>
<td>RETF</td>
<td>-</td>
<td>Far prosedürden dönüş yapar.

Algoritma:
<pre>Pop from stack:
IP
CS
if immediate operand is present: SP = SP + operand</pre>
<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td colspan="6">değişmez</td></tr></table></td>
</tr>
</table>

<table>
<tr>
<td>ROL</td>
<td>memory, immediate
REG, immediate
memory, CL
REG, CL</td>
<td>Birinci operandı sola döndürür. Bit olarak dönme sayısı ikinci operandla belirtilir.

Algoritma:
<pre>shift all bits left, the bit that goes off is set to CF and
the same bit is inserted to the right-most position.</pre>
Örnek:
<pre>MOV AL, 1Ch ; AL = 00011100b
ROL AL, 1 ; AL = 00111000b, CF=0.
RET</pre>
<table><tr><td>C</td><td>O</td></tr><tr><td>r</td><td>r</td></tr></table>
OF=0 olur eğer birinci operandı işaretini korursa.</td>
</tr>
<tr>
<td>ROR</td>
<td>memory, immediate
REG, immediate
memory, CL
REG, CL</td>
<td>Birinci operandı sağa döndürür. Bit olarak dönme sayısı ikinci operandla belirtilir.

Algoritma:
<pre>shift all bits right, the bit that goes off is set to CF
and the same bit is inserted to the left-most position.</pre>
Örnek:
<pre>MOV AL, 1Ch ; AL = 00011100b
ROR AL, 1 ; AL = 00001110b, CF=0.
RET</pre>
<table><tr><td>C</td><td>O</td></tr><tr><td>r</td><td>r</td></tr></table>
OF=0 olur eğer birinci operandı işaretini korursa.</td>
</tr>
<tr>
<td>SAHF</td>
<td>-</td>
<td>AH register’ına flag register’ının en az öneme sahip 8 bit’inin değerini yükler.

Algoritma:
<pre>flags register = AH
AH bit:7   6   5   4   3   2   1   0
      [SF] [ZF] [0] [AF] [0] [PF] [1] [CF]
1, 3, 5 bitler ayrılmıştır.</pre>
<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td></tr></table></td>
</tr>
</table>

<table>
<tr>
<td>SAL</td>
<td>memory, immediate
REG, immediate
memory, CL
REG, CL</td>
<td>Birinci operandı sola aritmetik kaydırır. Kaydırma sayısı ikinci operandla belirtilir.

Algoritma:
<pre>Shift all bits left, the bit that goes off is set to CF.
Zero bit is inserted to the right-most position.</pre>
Örnek:
<pre>MOV AL, 0E0h ; AL = 11100000b
SAL AL, 1 ; AL = 11000000b, CF=1.
RET</pre>
<table><tr><td>C</td><td>O</td></tr><tr><td>r</td><td>r</td></tr></table>
OF=0 eğer birinci operand işaretini korursa.</td>
</tr>
<tr>
<td>SAR</td>
<td>memory, immediate
REG, immediate
memory, CL
REG, CL</td>
<td>Birinci operandı sağa aritmetik kaydırır. Kaydırma sayısı ikinci operandla belirtilir.

Algoritma:
<pre>Shift all bits right, the bit that goes off is set to CF.
The sign bit that is inserted to the left-most position
has the same value as before shift.</pre>
Örnek:
<pre>MOV AL, 0E0h ; AL = 11100000b
SAR AL, 1 ; AL = 11110000b, CF=0.
MOV BL, 4Ch ; BL = 01001100b
SAR BL, 1 ; BL = 00100110b, CF=0.
RET</pre>
<table><tr><td>C</td><td>O</td></tr><tr><td>r</td><td>r</td></tr></table></td>
</tr>
<tr>
<td>SBB</td>
<td>REG, memory
memory, REG
REG, REG
memory, immediate
REG, immediate</td>
<td>Ödünç bitiyle çıkarma işlemi yapar.

Algoritma:
<pre>operand1 = operand1 - operand2 - CF</pre>
Örnek:
<pre>STC
MOV AL, 5
SBB AL, 3 ; AL = 5 - 3 - 1 = 1
RET</pre>
<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td></tr></table></td>
</tr>
</table>

<table>
<tr>
<td>SCASB</td>
<td>-</td>
<td>AL ile ES:[DI] adresindeki değeri karşılaştırır.

Algoritma:
<pre>ES:[DI] - AL
set flags according to result:
OF, SF, ZF, AF, PF, CF
if DF = 0 then
DI = DI + 1
else
DI = DI - 1</pre>
<table>
<tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr>
<tr><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td></tr>
</table>
</td>
</tr>
<tr>
<td>SCASW</td>
<td>-</td>
<td>AX ile ES:[DI] adresindeki değeri karşılaştırır.

Algoritma:
<pre>ES:[DI] - AX
set flags according to result:
OF, SF, ZF, AF, PF, CF
if DF = 0 then
DI = DI + 2
else
DI = DI - 2</pre>
<table>
<tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr>
<tr><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td></tr>
</table>
</td>
</tr>
<tr>
<td>SHL</td>
<td>memory, immediate
REG, immediate
memory, CL
REG, CL</td>
<td>Birinci operandı sola kaydırır. Kaydırma sayısı ikinci operanla belirtilir.

Algoritma:
<pre>Shift all bits left, the bit that goes off is set to CF.
Zero bit is inserted to the right-most position.</pre>
Örnek:
<pre>MOV AL, 11100000b
SHL AL, 1 ; AL = 11000000b, CF=1.
RET</pre>
<table>
<tr><td>C</td><td>O</td></tr>
<tr><td>r</td><td>r</td></tr>
</table>
OF=0 eğer birinci operand işaretini korursa.
</td>
</tr>
</table>

SHR	memory, immediate
	REG, immediate
	memory, CL
	REG, CL
		Birinci operandı sağa kaydırır. Kaydırma sayısı ikinci operanla belirtilir.
		
		Algoritma:
		`Shift all bits right, the bit that goes off is set to CF.`
		`Zero bit is inserted to the left-most position.`
		
		Örnek:
		`MOV AL, 00000111b`
		`SHR AL, 1 ; AL = 00000011b, CF=1.`
		`RET`
		
		C O
		r r
		
		OF=0 eğer birinci operand işaretini korursa.

STC	-	Taşma bayrağı (carry flag) set edilir. Algoritma: `CF = 1` C 1
STD	-	Yön bayrağı (direction flag) set edilir. CMPSB, CMPSW, LODSB, LODSW, MOVSB, MOVSW, STOSB, STOSW komutları tarafından SI ve DI azaltılır (D=1) veya artırılır (D=0). Algoritma: `DF = 1` D 1
STI	-	Kesme bayrağı (interrupt flag) set edilir. Bu donanım interrupt'larını etkin yapar. Algoritma: `IF = 1` I 1

<table>
<tr>
<td>STOSB</td>
<td>-</td>
<td>AL register’ının değerini ES:[DI] adresine kopyalar. DI güncellenir.

Algoritma:

```
ES:[DI] = AL
if DF = 0 then
DI = DI + 1
else
DI = DI - 1
```


Örnek:

```
#make_COM#
ORG 100h
LEA DI, a1
MOV AL, 12h
MOV CX, 5
REP STOSB
RET
a1 DB 5 dup(0)
```


<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td colspan="6">değişmez</td></tr></table></td>
</tr>
<tr>
<td>STOSW</td>
<td>-</td>
<td>AX register’ının değerini ES:[DI] adresine kopyalar. DI güncellenir.

Algoritma:

```
ES:[DI] = AX
if DF = 0 then
DI = DI + 2
else
DI = DI - 2
```


Örnek:

```
#make_COM#
ORG 100h
LEA DI, a1
MOV AX, 1234h
MOV CX, 5
REP STOSW
RET
a1 DW 5 dup(0)
```


<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td colspan="6">değişmez</td></tr></table></td>
</tr>
</table>

<table>
<tr>
<td>SUB</td>
<td>REG, memory
memory, REG
REG, REG
memory, immediate
REG, immediate</td>
<td>Çıkarma yapar.

Algoritma:
<code>operand1 = operand1 - operand2</code>

Örnek:
<pre>MOV AL, 5
SUB AL, 1 ; AL = 4
RET</pre>
<table>
<tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr>
<tr><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td><td>r</td></tr>
</table>
</td>
</tr>
<tr>
<td>TEST</td>
<td>REG, memory
memory, REG
REG, REG
memory, immediate
REG, immediate</td>
<td>İk operand arsında karşılıklı bitlerin hepsine e işlemi uygular. Operandların değeri değişmez ZF, SF ve PF bayrak bitleri değişir.

Doğruluk tablosu:
<pre>1 AND 1 = 1
1 AND 0 = 0
0 AND 1 = 0
0 AND 0 = 0</pre>
Örnek:
<pre>MOV AL, 00000101b
TEST AL, 1 ; ZF = 0.
TEST AL, 10b ; ZF = 1.
RET</pre>
<table>
<tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td></tr>
<tr><td>0</td><td>r</td><td>r</td><td>0</td><td>r</td></tr>
</table>
</td>
</tr>
<tr>
<td>XCHG</td>
<td>REG, memory
memory, REG
REG, REG</td>
<td>İki operandın değerlerini yer değiştirir.

Algoritma:
<code>operand1 < - > operand2</code>

Örnek:
<pre>MOV AL, 5
MOV AH, 2
XCHG AL, AH ; AL = 2, AH = 5
XCHG AL, AH ; AL = 5, AH = 2
RET</pre>
<table>
<tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr>
<tr><td colspan="6">değişmez</td></tr>
</table>
</td>
</tr>
</table>

<table>
<tr>
<td>XLATB</td>
<td>-</td>
<td>Tablo kullanarak bir byte çevirir. DS:[BX+işaretsiz AL] adresinden bir byte bilgiyi AL register’ına kopyalar.

Algoritma:
<pre>AL = DS:[BX + unsigned AL]</pre>
Örnek:
<pre>#make_COM#
ORG 100h
LEA BX, dat
MOV AL, 2
XLATB ; AL = 33h
RET
dat DB 11h, 22h, 33h, 44h, 55h</pre>
<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td colspan="6">değişmez</td></tr></table></td>
</tr>
<tr>
<td>XOR</td>
<td>REG, memory
memory, REG
REG, REG
memory, immediate
REG, immediate</td>
<td>İki operandın karşılıklı bitlerine XOR işlemi yapılır. Sonuç birinci operanda saklanır.

Doğruluk tablosu:
<pre>1 XOR 1 = 0
1 XOR 0 = 1
0 XOR 1 = 1
0 XOR 0 = 0</pre>
Örnek:
<pre>MOV AL, 00000111b
XOR AL, 00000010b ; AL = 00000101b
RET</pre>
<table><tr><td>C</td><td>Z</td><td>S</td><td>O</td><td>P</td><td>A</td></tr><tr><td>0</td><td>r</td><td>r</td><td>0</td><td>r</td><td>?</td></tr></table></td>
</tr>
</table>

ÖRNEKLER

```
; hafizada yeralan ve herbirisi 20 rakamdan olusan
; 16'lik tabandaki iki sayinin toplamini yapar.
; ilk sayi 0100h-0109h
; ikinci sayi 010Ah-0113h
; sonuc 0114h-011Dh
; m.ali akcayol
; 01.07.2007

org 0100h
mov [0100h],8A76h
mov [0102h],6557h
mov [0104h],1A98h
mov [0106h],713Eh
mov [0108h],8797h
mov [010Ah],2587h
mov [010Ch],8B96h
mov [010Eh],2588h
mov [0110h],46D1h
mov [0112h],9854h

mov si, 0108h           ; 1.sayinin en sagdaki 4 rakam (1 word) aliniyor
mov bx,[si]
add bx,[si+000Ah]       ; 2.sayinin en sagdaki 4 rakami ile toplaniyor
mov [si+0014h],bx       ; sonuc hafizaya yaziliyor
jnc atla1               ; carry=0 ise atla
    mov dx,1            ; sub isleminden once carry dx'e saklaniyor
    atla1:
sub si,2                ; sola dogru 4 rakam (1 word) gidiliyor
dongu:
    mov bx,[si]         ; 1.sayinin onceki 4 rakami (1 word) aliniyor
    add bx,[si+000Ah]   ; 2.sayinin onceki 4 rakami ile toplaniyor
    add bx,dx           ; varsa carry ekleniyor
    xor dx,dx           ; saklanan carry degeri sifirlaniyor
    mov [si+0014h],bx   ; sonuc hafizaya yaziliyor
    jnc atla2           ; carry=0 ise atla
      mov dl,1
    atla2:
        sub si,2        ; sola dogru 4 rakam (1 word) gidiliyor
        cmp si,0100h    ; en sola gelindimi?
        jae dongu       ; gelinmediyse basa doner
call mesajyaz
hlt

mesajyaz proc
    mov dx, offset mesajvar
    mov ah, 9
    int 21h
    ret
    mesajvar db "toplama islemi tamamlandi... $"
mesajyaz endp
```

```
; hafizada yeralan toplam 100 karakterden
; ADET ile belirtilen kadarini kendi
; yerinde siralayan algoritma - bubblesort
; m.ali akcayol
; 28.06.2007


org 0100h
dizi0 db 'v','a','z','b','t','r','z','k','m','a'
dizi1 db 'v','a','z','b','t','r','z','k','m','a'
dizi2 db 'v','a','z','b','t','r','z','k','m','a'
dizi3 db 'v','a','z','b','t','r','z','k','m','a'
dizi4 db 'v','a','z','b','t','r','z','k','m','a'
dizi5 db 'v','a','z','b','t','r','z','k','m','a'
dizi6 db 'v','a','z','b','t','r','z','k','m','a'
dizi7 db 'v','a','z','b','t','r','z','k','m','a'
dizi8 db 'v','a','z','b','t','r','z','k','m','a'
dizi9 db 'v','a','z','b','t','r','z','k','m','a'

ADET dw 9                   ; ilk 10 karakter siralanir

mov dx,ADET                 ; eleman sayisi-1
call ekranayazdir

dongu:                      ; toplam eleman sayisi kadar tekrar
    mov cx,0                ; siralanmamis eleman sayisi
    icdongu:                ; kalan eleman sayisi kadar tekrar
        mov si,cx
        mov al,[dizi0+si]   ; soldaki eleman
        mov bl,[dizi0+si+1] ; sagdaki eleman
        cmp al,bl
        jbe devam           ; soldaki kucukse degistrme yapma
        call degistir       ; soldaki buyukse degistir
        devam:
            inc cx          ; siralanmamis elemanlarda bir artir
            cmp cx,dx       ; siralanmamis elemanlarin sonuna geldimi
            jnb icdongusonu ; siralanmamis elemanlarin sonu
    jmp icdongu             ; siralanmamis eleman devam ediyor
    icdongusonu:            ; ic dongude siralanmamis eleman kalmadi
    dec dx                  ; sirasiz kalan eleman sayisini bir azalt
    cmp dx,0                ; sirasiz eleman sayisi 0 mi?
    je bitir                ; sirasiz eleman sayisi 0 ise bitir
    jmp dongu               ; sirasiz eleman varsa basa git
bitir:
call ekranayazdir           ; sirali yazdir
hlt                         ; program bitisi

degistir proc               ; yanyana iki karakter yer degistirir
    mov [dizi0+si],bl
    mov [dizi0+si+1],al
    ret
degistir endp

ekranayazdir proc
    push dx                 ; dl ye deger atanacak
    mov cx, 0               ; karakter sayaci
    mov ah, 2               ; int 21, ah=2, ekrana karakter yazdir
    tekrar:
```

```
        mov si,cx
        mov dl, [dizi0+si]
        int 21h             ; int 21, ah=2, ekrana karakter yazdir
        inc cx
        cmp cx, ADET
        ja cikis
    jmp tekrar
    cikis:
    mov dl, 0Dh             ; return
    int 21h
    mov dl, 0Ah             ; yeni satir
    int 21h
    pop dx                  ; dl nin eski degeri alindi
    ret
ekranayazdir endp
```

```
; binary search ---
; sirali bir dizide en cok log(2)sayiadedi
; kadar kontrolle aranan sayinin olup
; olmadigini bulur
; m.ali akcayol
; 29.06.2007


org 0100h
mov [0100h], 1
mov [0101h], 2
mov [0102h], 3
mov [0103h], 4
mov [0104h], 6
mov [0105h], 8
mov [0106h], 9
mov [0107h], 10
mov [0108h], 11
mov [0109h], 13
mov [010Ah], 14
mov [010Bh], 15
mov [010Ch], 16
mov [010Dh], 17
mov [010Eh], 19
mov [010Fh], 20
mov [0110h], 22
mov [0111h], 23
mov [0112h], 25
mov [0113h], 27
mov [0114h], 28
mov [0115h], 30

mov dl,2        ; bolen
mov dh,28       ; aranan sayi
mov bx,0100h    ; en kucuk adres
mov cx,0115h    ; en buyuk adres
mov si,0100h

mov ax,cx       ; eleman sayisi hesaplaniyor
dec bx
sub ax,bx
div dl          ; ortadaki sayi bulunuyor
xor ah,ah       ; kalan atiliyor
add si,ax       ; ortadaki sayinin adresi
dongu:
    cmp [si],dh     ; ortadaki sayi arananla karsilastiriliyor
    jb sagtaraf     ; kucukse saga gidilir
    ja soltaraf     ; buyukse sola gidilir
    je bulundu      ; esitse bulunmustur
    sagtaraf:
        mov bx,si
        mov ax,cx       ; eleman sayisi hesaplaniyor
        sub ax,bx
        div dl          ; ortadaki sayi bulunuyor
        cmp al,0
        jz kalanal1
        xor ah,ah       ; kalan atiliyor
        add si,ax       ; ortadaki sayinin adresi
```

```
        jmp dongu       ; basa don
        kalanal1:       ; son kalan eleman kontrol ediliyor
            xchg al,ah
            add si,ax
            cmp [si],dh
            je bulundu
            jne sayiyok
    soltaraf:
        mov cx,si
        mov ax,cx       ; eleman sayisi hesaplaniyor
        sub ax,bx
        div dl          ; ortadaki sayi bulunuyor
        cmp al,0
        jz kalanal2
        xor ah,ah       ; kalan atiliyor
        sub si,ax       ; ortadaki sayinin adresi
        jmp dongu       ; basa don
        kalanal2:       ; son kalan eleman kontrol ediliyor
            xchg al,ah
            sub si,ax
            cmp [si],dh
            je bulundu
            jne sayiyok

; yazdirma islemleri

bulundu:
    mov dx, offset mesajvar
    mov ah, 9
    int 21h
    ret
    mesajvar db "sayi bulundu $"
    jmp cikis
sayiyok:
    mov dx, offset mesajyok
    mov ah, 9
    int 21h
    ret
    mesajyok db "sayi yok $"
cikis:
hlt
```