

زبان ماشین و اسمبلی

مدرس: سیاوش خجسته

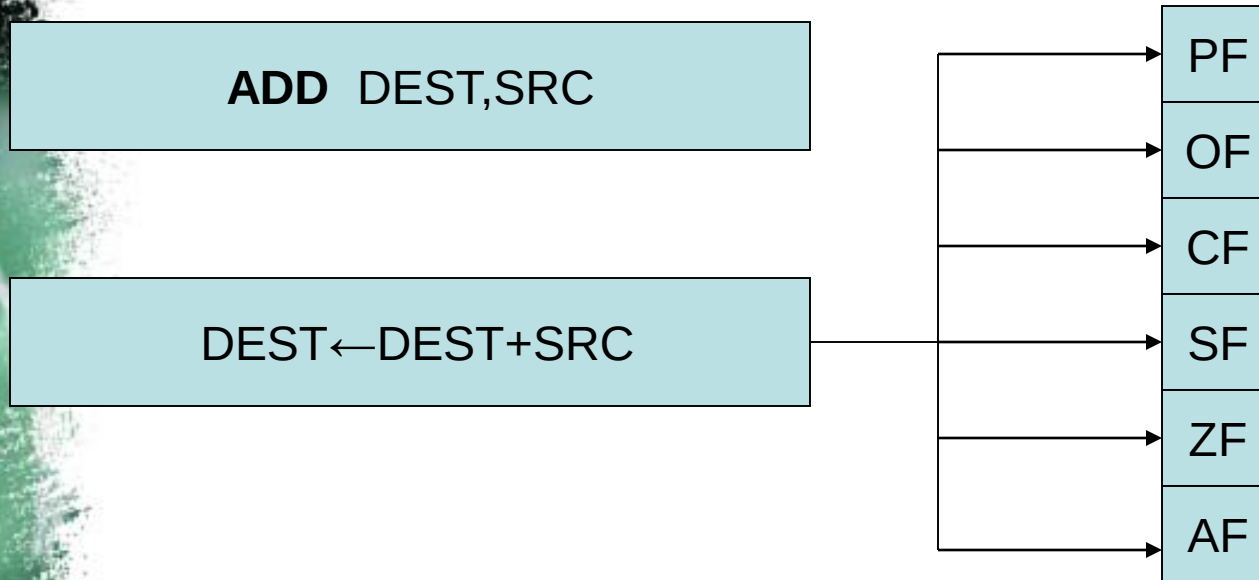
www.samavi.info

دستورات زبان اسمبلی

دستورات محاسباتی

- جمع
- جمع به کمک بیت نقلی
- تفریق
- تفریق با بیت قرضی
- گسترش بایت به کلمه
- گسترش کلمه به LONG
- ضرب
- تقسیم
- منفی کردن
- کاهش
- افزایش
- جمع BCD
- تفریق BCD
- ...

جمع (ADD)



جمع (ADD)

CORRECT

ADD AX, BX

$AX \leftarrow AX + BX$

ADD AX, M

$AX \leftarrow AX + M$

ADD M, AX

$M \leftarrow M + AX$

ADD CX, 0FC25H

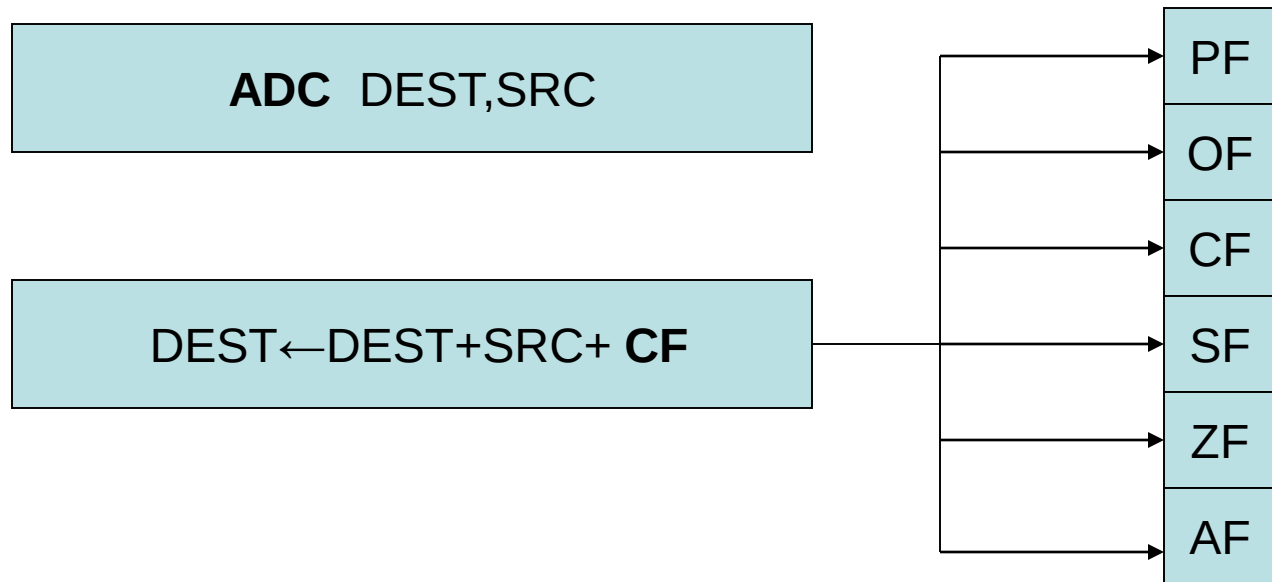
$CX \leftarrow CX + 0FC25H$

ERROR

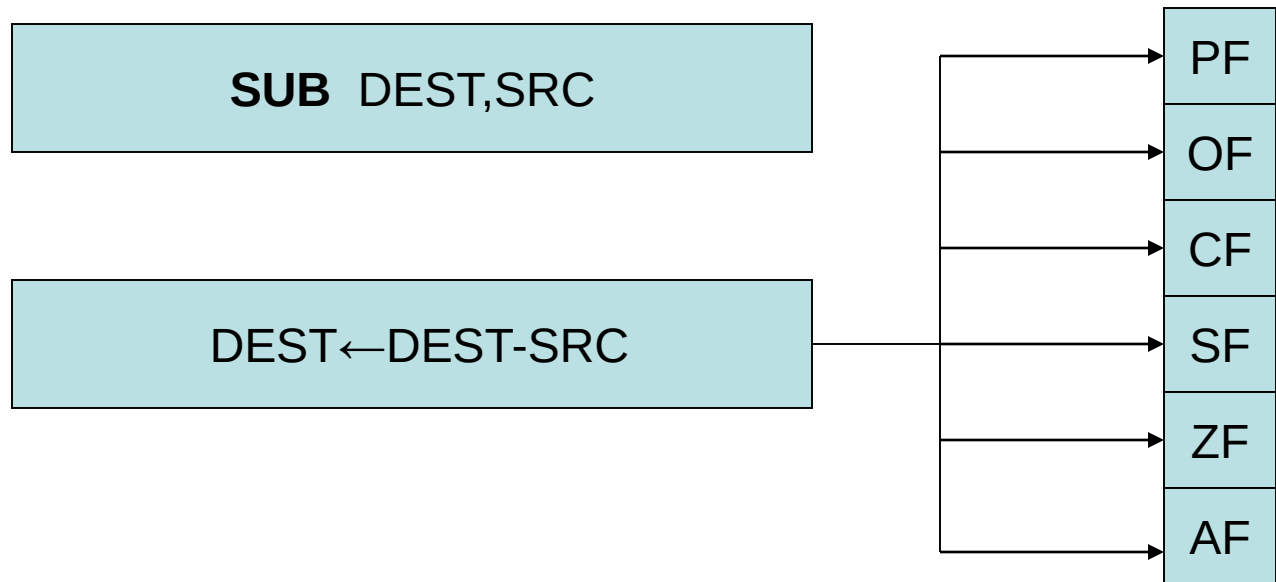
~~ADD M, N~~

~~ADD AL, CX~~

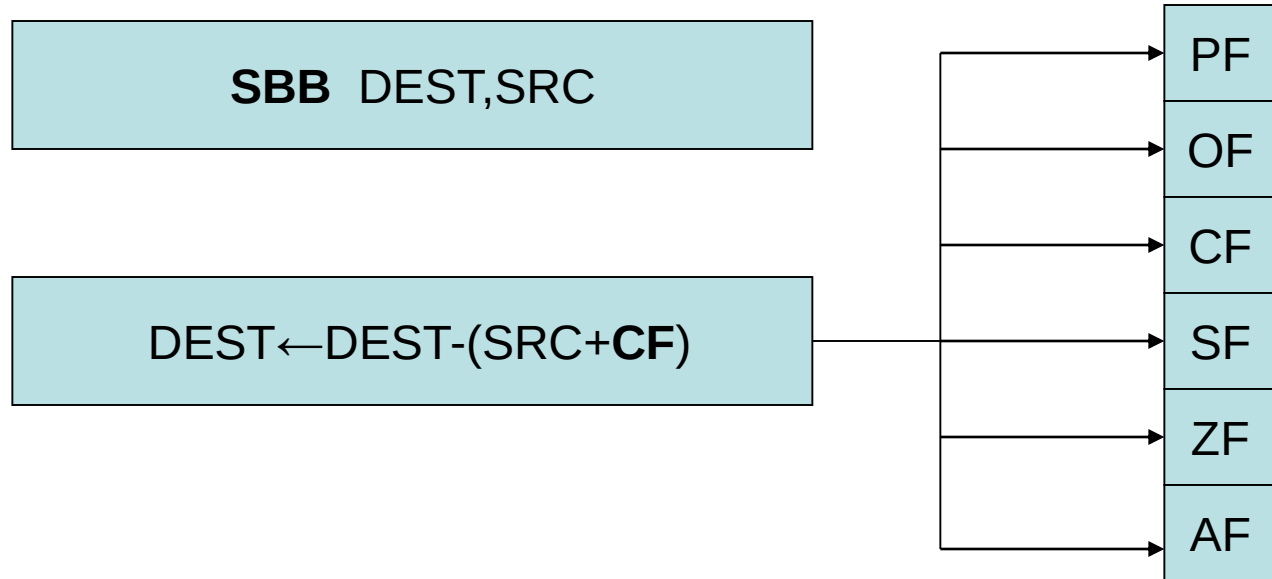
جمع با بیت نقلی (ADC)



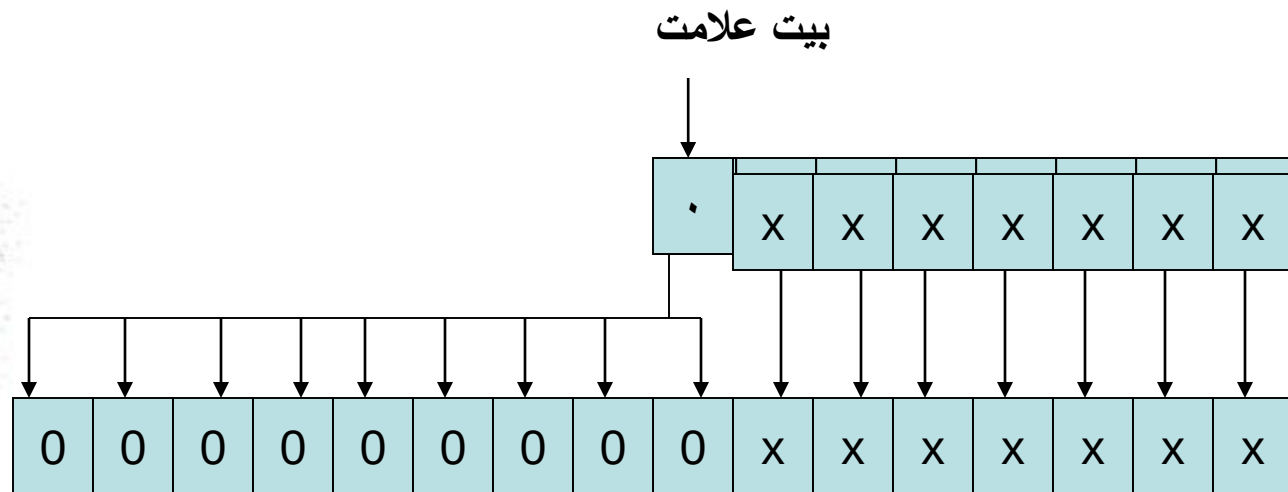
تفریق (SUB)



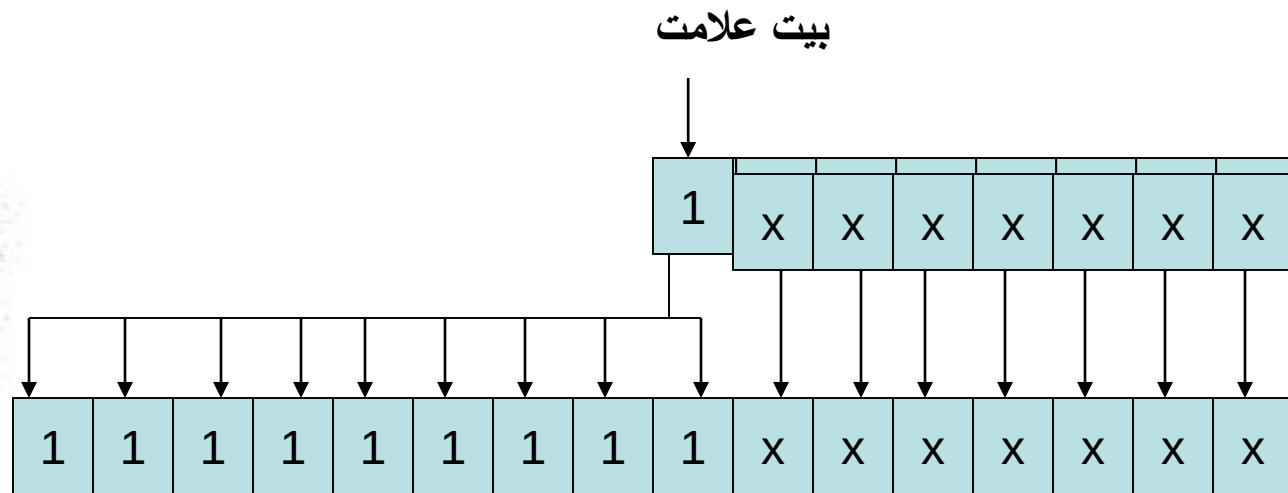
تفریق به کمک بیت قرضی (SBB)



گسترش BYTE به WORD



گسترش byte به word



دستورات گسترش داده

Convert Byte to Word

CBW •

$AL \rightarrow AX$

Convert Word to double

CWD •

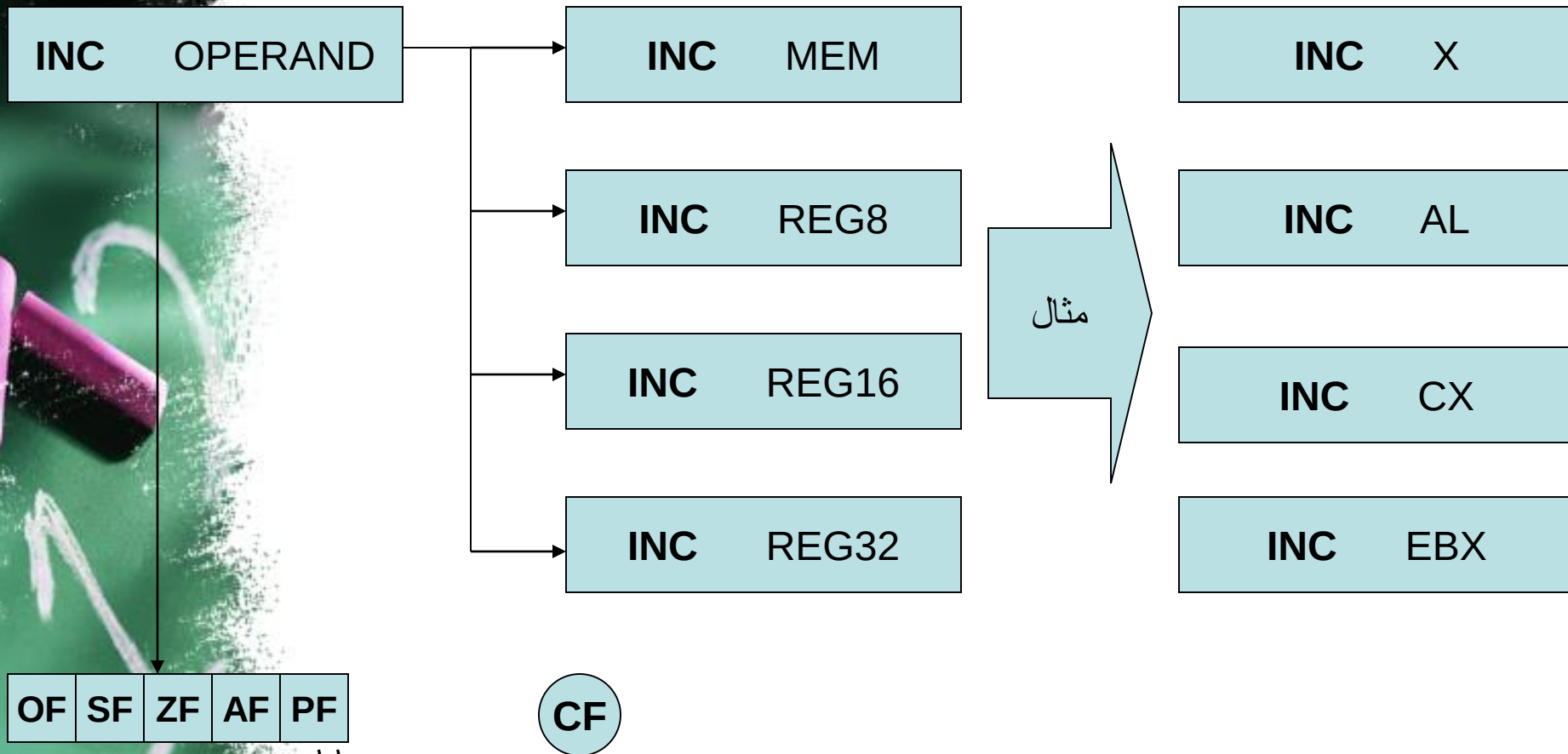
$AX \rightarrow DX, AX$

Convert Word to extended double word

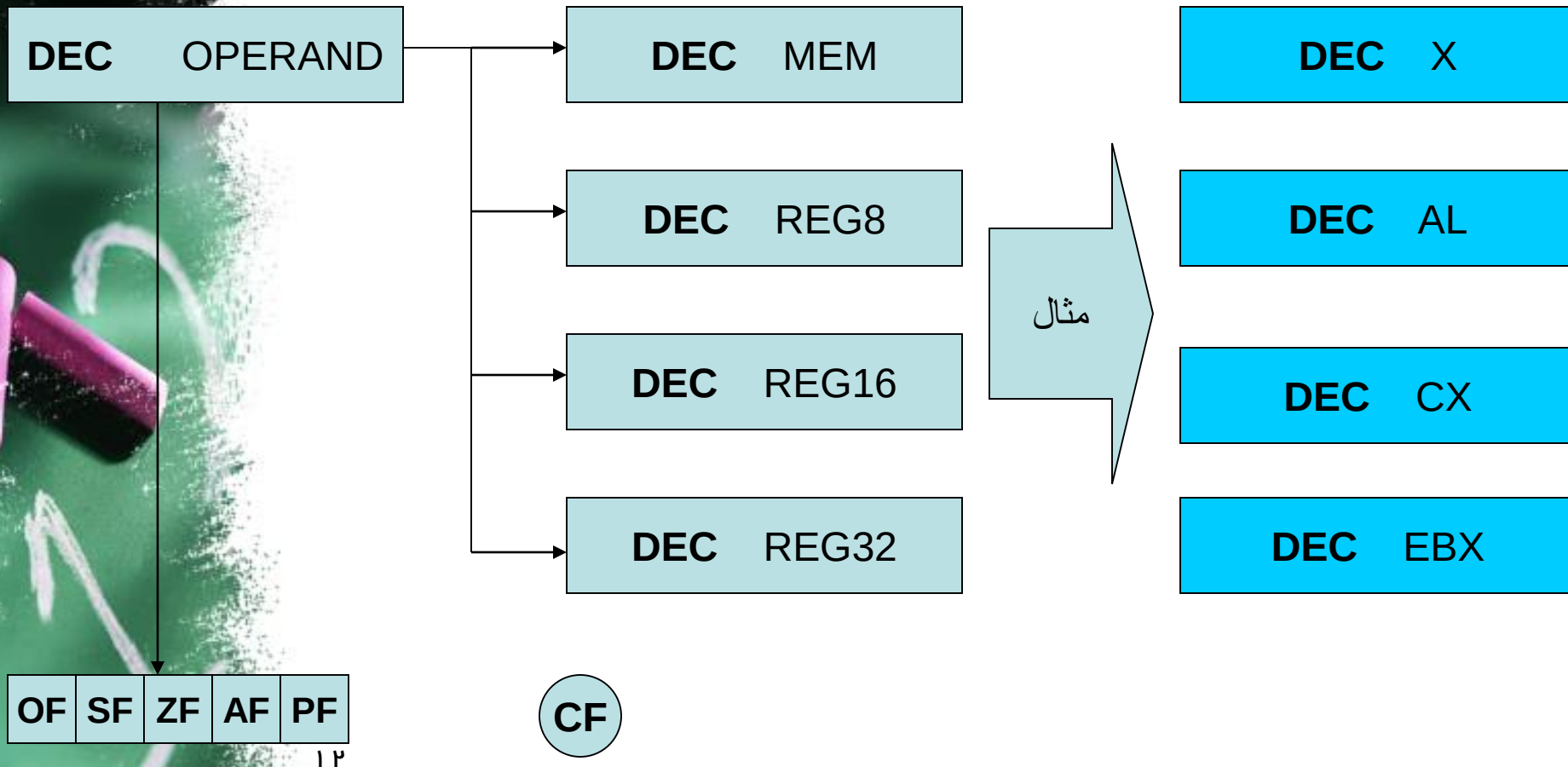
CWDE •

$AX \rightarrow EAX$

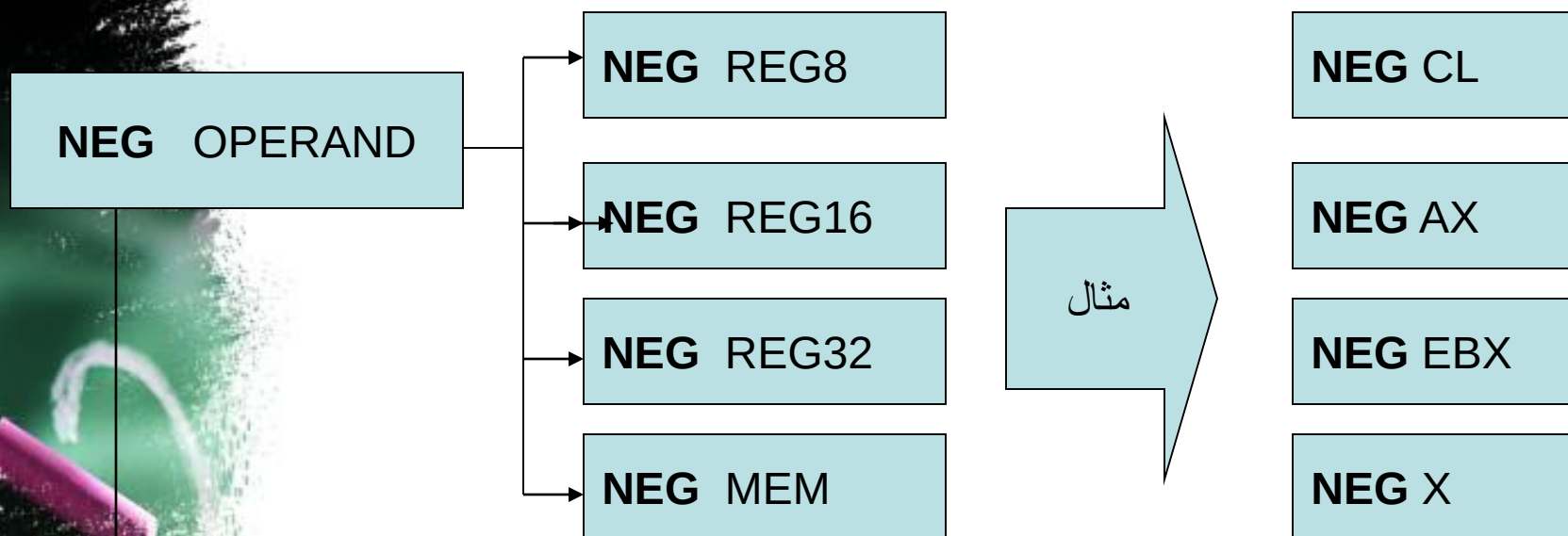
افزایش INCREMENT



کاهش DECREMENT



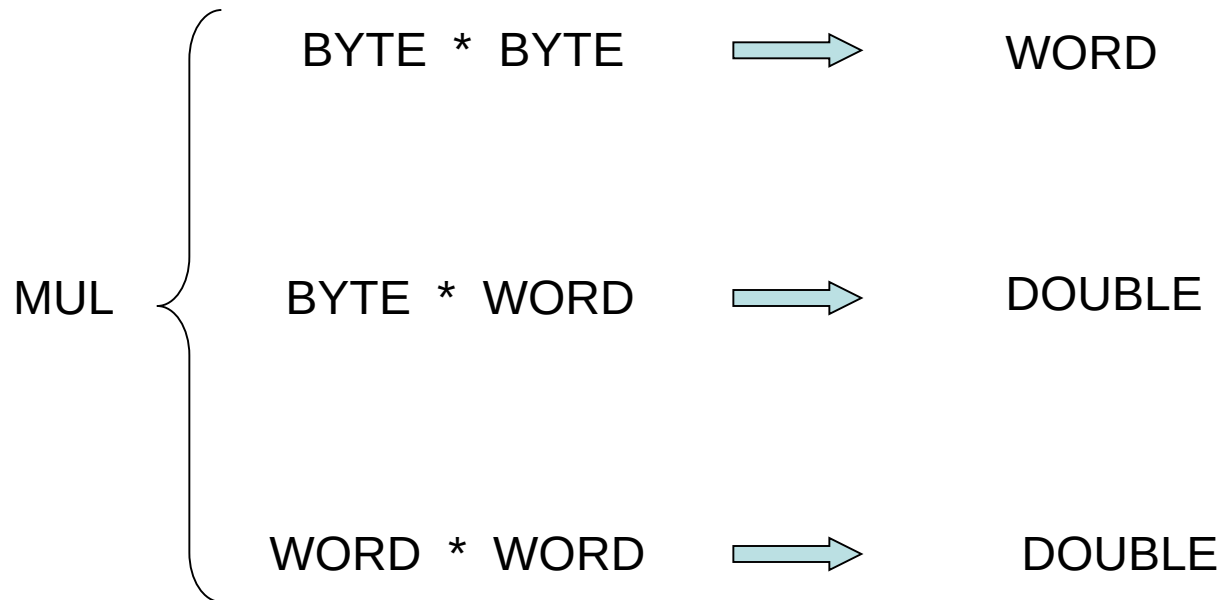
قرینه کردن NEGATIVE



OF	SF	ZF	AF	PF	CF
----	----	----	----	----	----

قرینه کردن معادل با مکمل
دو عدد است

ضرب (MUL)



MUL

BYTE * BYTE $\longrightarrow$ WORD

MUL OPERAND

$AX \leftarrow \text{OPERAND} * AL$

مثال

```
MOV AL,100  
MOV BL,200  
MUL BL
```



$AX=200*100=20000$

```
MOV AL,100  
MUL 55
```



$AX=55*100=5500$

در ضرب بایت در بایت همیشه یکی از اپرند ها AL است

MUL

WORD * WORD $\longrightarrow$ DOUBLE

MUL OPERAND

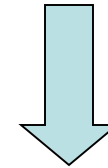
DX	AX	$\leftarrow$ OPERAND * AX
----	----	----------------------------------

مثال

```
MOV AX,1000  
MOV CX,5000  
MUL CX
```



$(DX,AX) \leftarrow AX * CX$



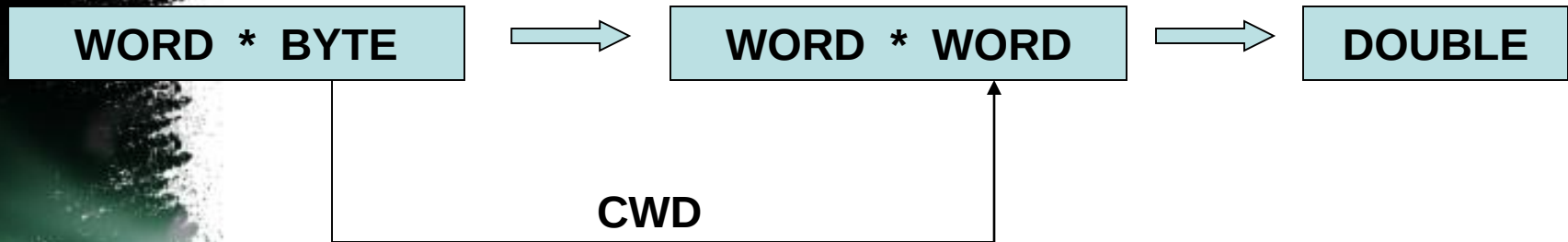
DX

4C

AX

4B40

MUL



```
MOV AL,100  
CBW  
MUL BX
```

```
MOV AX,0064H  
MUL BX
```

```
(DX,AX) ← AX * BX
```

ضرب

همیشه یکی از ابراند ها در AL یا AX قرار دارد

- ضرب اعداد بی علامت (UNSIGNED)

MUL

- ضرب علامتدار (SIGNED)

IMUL

نتیجه محاسبات در AX یا در (DX,AX) قرار می گیرد

DIVISION تقسيم

DIV

WORD ÷ BYTE

AX ÷ byte

DOUBLE ÷ WORD

(DX AX) ÷ word

DIV

DIV OPERAND8

AX

OPERAND8

AL

AH

DIV OPERAND16

DX,AX

OPERAND16

AX

DX

مثال

```
MOV AX,102  
MOV CL,5  
DIV CL
```



**$AL = AX / CL = 20$
 $AH = AX \% CL = 2$**

```
MOV DX,4040H  
MOV AX,1200H  
MOV BX,500H  
DIV BX
```



**$(DX,AX) = 40401200H$
 $AX = (DX,AX) / BX$
 $DX = (DX,AX) \% BX$**

```
MOV AX, 1000  
DIV 120
```



ERROR

عملوند تقسیم نمی تواند
عدد باشد

تقسیم

- علامتدار

idiv

همواره عملوند اول تقسیم
در (DX,AX) یا AX قرار
دارد

- بی علامت

div

خارج قسمت در AX یا AL
قرار می گیرد و باقیمانده در
AH یا DX قرار میگیرد

دستور XCHG

- برای مبادله داده استفاده می شود

XCHG dst,src

پس از اجرای دستور محتوای SRC و dst با هم مبادله می شوند.

۱. هیچ یک از طرفین نباید ثابت باشد
۲. هر دو نمی تواند متغیر باشد
۳. بر روی فلگ ها اثری ندارد
۴. طرفین باید هم اندازه باشند

دستور العمل LEA

این دستور العمل مخفف کلمات **Load effect address** می باشد.
شکل کلی دستور العمل بصورت زیر می باشد :

LEA destination, source

destination بایستی یک ثبات ۱۶ بیتی بوده و **source** هر گونه رجوعی به حافظه می باشد. این دستور العمل آدرس **source** را در **destination** قرار می دهد.

مثال :

LEA BX , X

آدرس متغیر **X** در ثبات **BX** قرار می گیرد.

این دستور العمل معادل دستور العمل زیر می باشد.

MOV BX , OFFSET



دستورات کتترلی

- انشعاب غیر شرطی
– مشابه دستور GOTO
- انشعاب شرطی
– مشابه با IF



JMP

دستور **JMP** شبیه **goto** در پاسکال می باشد. این دستور دارای فرم زیر است :

JMP LABEL



```
MOV AX , 100
L1 : INC AX
ADD AX,BX
JMP L1
```

The diagram shows a light blue rectangular box containing assembly code. To the left of the box, a white line forms a loop, starting from the 'JMP L1' instruction, going left, then up, then right, and ending with an arrow pointing to the 'L1 : INC AX' instruction. This illustrates the jump back to the start of the loop.

حلقه بی نهایت

CMP

• مقایسه دو عدد (COMPARE)

دستورالعمل **CMP** مانند دستورالعمل **SUB** عمل نموده ولی نتایج در جایی ذخیره نمی شود بلکه محتوی فلگ ها را تغییر می دهد.

CMP OP1,OP2

OP1-OP2

OP1 و OP2 تغییر نمی کنند

OF

SF

ZF

AF

DF

CF

CF

ZF

OP1=OP2

0

1

OP1>OP2

0

0

OP1<OP2

1

0



```
CMP AX,BX
```

```
CMP AX, Z
```

```
CMP Z, AX
```

```
CMP AX,120
```

CORRECT USE

```
CMP 120,Y
```

```
CMP 200, CX
```

ERROR

حلقه تکرار در زبان اسمبلی

در حلقه تکرار **FOR** در همه زبان های برنامه سازی ،تعداد دفعاتی که بدنه حلقه باید اجرا شود از قبل معین می باشد. در زبان اسمبلی این تعداد را بایستی در ثبات **CX** قرار داد و دستورالعمل تکرار دستورالعمل **LOOP** می باشد.

شکل کلی عبارتست از:

LOOP **Statement _ label**



LOOP

- هر بار یک واحد از CX کم می کند
- پرش زمانی انجام میشود که CX صفر نباشد
- برای ایجاد حلقه FOR مناسب است



```
MOV AX,0  
MOV X,AX  
MOV CX,20  
FOR : MOV AX,X  
INC X  
INC AX  
LOOP FOR
```

```
X=0;  
For(i=0;i<20;i++)  
X++;
```

LOOPZ

- حلقه تا زمانی که CX مخالف صفر و ZF برابر یک باشد تکرار می شود

```
I=10;  
WHILE ( I # 0) AND (X=1200)  
{  
X=X+Y  
I--;  
}
```

```
MOV CX,10  
LABEL : MOV AX,X  
ADD AX,Y  
MOV X,AX  
CMP AX,1200  
LOOPZ LABEL
```

LOOPZ علاوه بر CX ، ZF را نیز بررسی می کند

بلافاصله با غیر صفر شدن نتیجه یک محاسبه (در مثال فوق در دستور cmp) ZF=0 شده و از حلقه خارج میشود

```
WHILE ( CX#0) AND ( ZF=1)  
{  
}
```

- دستور LOOPE دقیقاً مثل LOOPZ است

LOOPNZ

- تکرار تا زمانی که :

– CX صفر نشده

– نتیجه محاسبات غیر صفر شود

- خروج از حلقه

– CX صفر شود

– نتیجه محاسبه صفر شود (تساوی)

```
WHILE(CX#0) AND (ZF=0)
{
}
```

دستور LOOPNE هم به همین معنی بکار برده میشود

مثال :

```
MOV      CX , 10
FOR :
      .
      .
      .
CMP      BX , 0
LOOPNE   FOR
```



دستورات پرش شرطی

- این دستورات از فلگ ها اثر می پذیرند
- قبل از این دستورات پردازش لازم جهت تغییر فلگ ها انجام میشود

مبتنی بر فلگ ها

برای اعداد علامتدار

برای اعداد بی علامت

دستورات پرشی



پرش مبتنی بر فلگ ها

JUMP SIGN (NEGATIVE)	SF=1	JS
JUMP NOT SIGN (POSITIVE)	SF=0	JNS
JUMP CARRY	CF=1	JC
JUMP NOT CARRY	CF=0	JNC
JUMP OVERFLOW	OF=1	JO
JUMP NOT OVERFLOW	OF=0	JNO
JUMP PARITY	PF=1	JP
JUMP NOT PARITY	PF=0	JNP

پرش برای اعداد بی علامت

JUMP EQUAL

JE

OP1=OP2

JUMP NOT EQUAL

JNE

OP1#OP2

JUMP ABOVE

JA

OP1>OP2

JUM ABOVE OR EQUAL

JAE

OP1>=OP2

JUMP BELOW

JB

OP1<OP2

JUMP BELOW OR EQUAL

JBE

OP1<=OP2

پرش برای اعداد علامتدار

JUMP EQUAL	JE	OP1=OP2
JUMP NOT EQUAL	JNE	OP1#OP2
JUMP GREATER	JG	OP1>OP2
JUMP GREATER OR EQUAL	JGE	OP1>=OP2
JUMP LESS	JL	OP1<OP2
JUMP LESS OR EQUAL	JLE	OP1<=OP2

مثال :

JZ END _ WHILE

این دستورالعمل بدین معنی است که اگر فلگ ZF برابر، یک باشد کنترل به دستورالعمل با برچسب END _ WHILE منتقل می گردد در غیر این صورت کنترل به دستورالعمل بعدی می رود.



نکته :

پس از دستور **CMP** در صورتی که عملوندها بدون علامت در نظر گرفته شوند از دستورات عملهای پرش شرطی زیر می توان استفاده نمود :

نام دستورالعمل	معنی	فلگها برای پرش
Ja	پرش در حالت بالاتر	CF=0,ZF=0
Jnbe	پرش در حالت پایین یا مساوی	
Jae	پرش در حالت بالاتر یا مساوی	CF=0
Jnb	پرش در حالت پایین تر نبودن	
Jb	پرش در حالت پایین تر	CF=1
Jnae	پرش در حالت پایین تر یا مساوی نبودن	
Jbe	پرش در حالت پایین تر یا مساوی	ZF=1 یا CF=1
Jna	پرش در حالت بالاتر نبودن	



نکته :

پس از دستور **CMP** در صورتیکه عملوندها با علامت در نظر گرفته شوند از دستورالعملهای پرش شرطی زیر می توان استفاده نمود :

نام دستورالعمل	معنی	فلگها برای پرش
Jg	پرش در حالت بزرگتر	SF=OF,ZF=0
Jnle	پرش در حالت کوچکتر یا مساوی نبودن	
Jge	پرش در حالت بزرگتر یا مساوی	SF=OF
Jnl	پرش در حالت کوچکتر نبودن	
Jl	پرش در حالت کوچکتر	SF<>OF
Jnge	پرش در حالت بزرگتر یا مساوی نبودن	
Jle	پرش در حالت کوچکتر یا مساوی	SF<>OF یا ZF =1
Jna	پرش در حالت بزرگتر نبودن	



دستور العمل JCXZ

دستور العمل JCXZ یک نوع پرش می باشد. منتهی پرش روی فلگی انجام نمی شود بلکه چنانچه مقدار ثبات CX برابر با صفر باشد پرش انجام می شود. شکل کلی بصورت زیر می باشد :

JCXZ Statement _ label



مثال :

```
MOV      CX , 50
LABI:    .
        .
        .
        DEC      CX
        JCXZ     LABEND
        JMP      LABI
```

LABEND:

دستورالعملهای فوق باعث میشود که بدنه دستورالعمل تکرار ۵۰ بار اجرا گردد.



