

بسمه تعالی



زبان ماشین و اسمبلی

علی چوداری خسروشاهی

Akhosroshahi@iaut.ac.ir

دانشگاه آزاد اسلامی

1

مرجع

IBM PC Assembly Language and Programing

برنامه نویسی و زبان اسمبلی کامپیوترهای شخصی

نویسنده: Peter Abel

2

فصل اول

مبانی اعداد

3

مبانی اعداد

- در زبانهای سطح بالا نگران اینکه داده ها در کامپیوتر چگونه نمایش داده میشوند نیستیم ولی در زبان های اسمبلی بایستی بفکر چگونگی ذخیره داده باشیم و اغلب با کار تبدیل داده ها از یک نوع به نوع دیگر مواجه می باشیم.
- یک عدد در مبنای M از ارقام $0, 1, \dots, M-1$ تشکیل می شود.
- حافظه های کامپیوتر فقط می تواند ارقام 0 یا 1 را در خود ذخیره نماید که به آنها بیت گفته میشود. در سیستم دودویی اعداد از بیت ها تشکیل شده اند.
- از جمله پر کاربرد ترین مبنای عددی عبارتند از ۲، ۸، ۱۰ و ۱۶

مبنا	ارقام تشکیل دهنده
2 (binary)	0, 1
8 (Octet)	0, 1, 2, 3, 4, 5, 6, 7
10 (Decimal)	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
16 (Hexadecimal)	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

4

مقایسه اعداد در چهار مبنا

Hexadecimal	Octal	Binary	Decimal
0	00	0000	00
1	01	0001	01
2	02	0010	02
3	03	0011	03
4	04	0100	04
5	05	0101	05
6	06	0110	06
7	07	0111	07
8	10	1000	08
9	11	1001	09
A	12	1010	10
B	13	1011	11
C	14	1100	12
D	15	1101	13
E	16	1110	14
F	17	1111	15

5

ارزش ارقام

- در زمان قرار گرفتن در یک عدد، هر رقم دارای دو ارزش متفاوت است
 - ارزش مطلق
 - ارزش مکانی
- 6539
-
- ارزش مکانی هر رقم برابر است با :
 - ارزش مطلق ضرب در مبنا به توان شماره مکان
 - که شماره مکان از سمت راست ترین رقم و با مکان صفر شروع میشود

6

ارزش ارقام

■ مثال: ارزش مکانی رقم ۲ را مشخص کنید؟

$$\begin{array}{r} 2 \ 1 \ 0 \\ (627)_8 \end{array}$$

→ $2 \times 8^1 = 16$

■ ارزش مکانی هر کدام از ارقام عدد زیر را بدست آورید:

$$\begin{array}{r} 2 \ 1 \ 0 \\ (A2C)_{16} \end{array}$$

→ $12 \times 16^0 = 12$
→ $2 \times 16^1 = 32$
→ $10 \times 16^2 = 2560$

7

تبدیل مبنای ۱۰ به ۲

■ روش ۱: تقسیمات متوالی بر ۲

$$(10) \rightarrow (2)$$

$$\begin{array}{r} 58 \quad 2 \\ \hline 58 \quad 29 \quad 2 \\ \hline 0 \quad 28 \quad 14 \quad 2 \\ \hline \quad 1 \quad 14 \quad 7 \quad 2 \\ \hline \quad \quad 0 \quad 6 \quad 3 \quad 2 \\ \hline \quad \quad \quad 1 \quad 2 \quad 1 \quad 2 \\ \hline \quad \quad \quad \quad 1 \quad 0 \quad 0 \\ \hline \quad \quad \quad \quad \quad 1 \end{array}$$

$$(58)_{10} = (111010)_2$$

8

تبدیل مبنای ۱۰ به ۲

■ روش دوم:

■ بزرگترین توان ۲ کوچکتر مساوی عدد را پیدا کرده و از آن کم می کنیم.

■ توان های ۲ عبارتند از: ۱، ۲، ۴، ۸، ۱۶، ۳۲، ...

■ این کار را تا زمانی که به صفر برسیم ادامه خواهیم داد.

■ بترتیب زیر توانهای ۲ استفاده شده ۱ و استفاده نشده ۰ قرار خواهیم داد.

$$\begin{array}{r} 58 \\ -32 \\ \hline 26 \\ -16 \\ \hline 10 \\ -8 \\ \hline 2 \\ -2 \\ \hline 0 \end{array}$$

$$\begin{array}{r} 32 \ 16 \ 8 \ 4 \ 2 \ 1 \\ 1 \ 1 \ 1 \ 0 \ 1 \ 0 \\ (58)_{10} = (111010)_2 \end{array}$$

در مثال فوق از عدد ۵۸ توانهای دوی ۲، ۴، ۸، ۱۶ و ۳۲ کم شده است پس زیر آنها ۱ و چون ۱۶ و ۴ استفاده نشده است ۰ قرار می دهیم.

9

تبدیل اعداد اعشاری مبنای ۱۰ به مبنای ۲

- اعداد را به دو قسمت صحیح و کسری تقسیم می کنیم و هر قسمت را جداگانه تبدیل می کنیم
- تبدیل قسمت صحیح به مبنای ۲.
- با تقسیمات متوالی بر ۲ و گرد آوری باقیمانده ها به عنوان رقم های مبنای ۲.
- تبدیل قسمت کسری به مبنای ۲.
- با ضرب متوالی در ۲ و گرد آوری اعداد صحیح تولید شده به عنوان رقم های مبنای جدید ۲.

10

تبدیل اعداد اعشاری مبنای ۱۰ به مبنای ۲

- مثال: عدد 41.6875 در مبنای ۱۰ می باشد. آنرا به مبنای ۲ تبدیل کنید.
- حل: ابتدا قسمت اعشار را از صحیح جدا کرده عمل تبدیل را برای هر یک جداگانه انجام می دهیم.

صحیح = 41

41	
20	1
10	0
5	0
2	1
1	0
0	1

 $(41)_{10} = (101001)_2$

اعشار = 0.6875

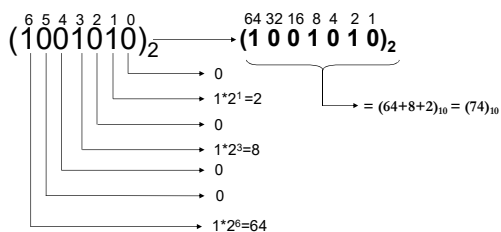
0.6875	
x 2	
1.3750	
x 2	
0.7500	
x 2	
1.5000	
x 2	
1.0000	

 $(0.6875)_{10} = (0.1011)_2$ $(41.6875)_{10} = (101001.1011)_2$

11

تبدیل مبنای ۲ به ۱۰

- محاسبه مجموع ارزش مکانی ارقام عدد



12

تبدیل مبنای ۲ به ۸

8 ← 2

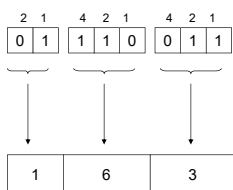
- کل ارقام عدد را با شروع از سمت راست ترین عدد به گروه های ۳ رقمی تقسیم می کنیم.
- اگر سمت چپ ترین گروه کمتر از سه تا شد می توان به تعداد مورد نیاز به سمت چپ آن صفر اضافه کرد.
- بترتیب معادل هر گروه را در مبنای ۸ می نویسیم.
- مثال: عدد $(101000110)_2$ را به مبنای ۸ تبدیل کنید؟

$$\left(\begin{array}{ccc} 101 & 000 & 110 \\ 5 & 8 & 8 \end{array} \right)_2 = (506)_8$$

13

تبدیل مبنای ۲ به ۸

- مثال: عدد $(01110011)_2$ را به مبنای ۸ تبدیل کنید



14

تبدیل مبنای ۸ به ۲

8 → 2

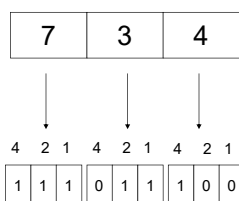
- معادل هر یک از ارقام را در مبنای ۲ و در میدان ۳ رقمی می نویسیم.
- اگر طول عدد کمتر از ۳ باشد به تعداد مورد نیاز صفر در سمت چپ اضافه می کنیم.
- گروه ها را بترتیب کنار هم قرار می دهیم.
- مثال: عدد $(517)_8$ را به مبنای ۲ تبدیل کنید؟

$$\left(\begin{array}{ccc} 5 & 1 & 7 \\ 101001111 \end{array} \right)_8 = (101001111)_2$$

15

تبدیل مبنای ۸ به ۲

■ مثال: عدد $(734)_8$ را به مبنای ۲ تبدیل نمایید



16

تبدیل مبنای ۲ به ۱۶

② ← ①6

■ کل ارقام عدد را با شروع از سمت راست ترین عدد به گروه های ۴ رقمی تقسیم می کنیم.

■ اگر سمت چپ ترین گروه کمتر از چهار تا شد می توان به تعدد مورد نیاز به سمت چپ آن صفر اضافه کرد.

■ بترتیب معادل هر گروه را در مبنای ۱۶ می نویسیم.

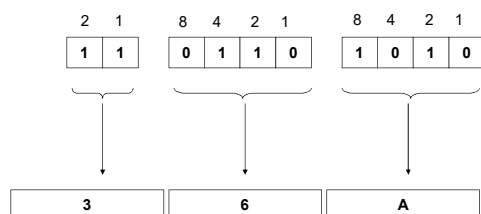
■ مثال: عدد $(101001110)_2$ را به مبنای ۱۶ تبدیل کنید؟

$$\left(\begin{array}{cccc} 000 & 1010 & 0111 & 0 \\ \hline & 1 & 4 & E \\ \hline \end{array} \right)_2 = (14E)_{16}$$

17

تبدیل مبنای ۲ به ۱۶

■ عدد $(1101101010)_2$ را به مبنای ۱۶ تبدیل کنید:



18

تبدیل مبنای ۱۶ به ۲

16 → 2

- معدل هر یک از ارقام را در مبنای ۲ و در میدان ۴ رقمی می نویسیم.
- اگر طول عدد کمتر از ۴ باشد به تعداد مورد نیاز صفر در سمت چپ اضافه می کنیم
- گروه ها را بترتیب کنار هم قرار می دهیم.

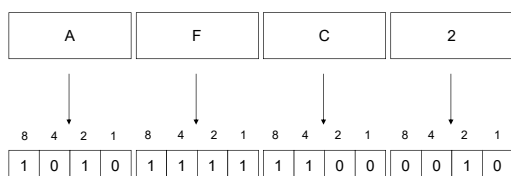
■ مثال: عدد $(2A5)_{16}$ را به مبنای ۲ تبدیل کنید؟

$$\left(\begin{array}{c} 2 \quad A \quad 5 \\ \hline 001010100101 \end{array} \right)_{16} = (001010100101)_2$$

19

تبدیل مبنای ۱۶ به ۲

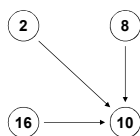
■ مثال: عدد $(AFC2)_{16}$ را به مبنای ۲ تبدیل نمایید:



20

تبدیل همه مبنایا به مبنای ۱۰

■ محاسبه مجموع ارزش مکانی ارقام عدد



■ مثال: عدد $(A2C)_{16}$ را به مبنای ۱۰ تبدیل کنید؟

$$(A2C)_{16} = 12 + 32 + 2560 = (2604)_{10}$$

21

سایر تبدیلات

■ برخی دیگر از تبدیلات و راه حل آنها.

تبدیل (مینا)	راه حل
۸ به ۱۰	تبدیل ۱۰ به ۲ سپس ۲ به ۸
۱۰ به ۸	تبدیل ۸ به ۲ سپس ۲ به ۱۰
۱۶ به ۱۰	تبدیل ۱۰ به ۲ سپس ۲ به ۱۶
۱۰ به ۱۶	تبدیل ۱۶ به ۲ سپس ۲ به ۱۰
۸ به ۱۶	تبدیل ۱۶ به ۲ سپس ۲ به ۸
۱۶ به ۸	تبدیل ۸ به ۲ سپس ۲ به ۱۶

22

سایر تبدیلات

■ مثال: عدد $(AC2)_{16}$ را به مبنای ۸ تبدیل نمایید.

حل (ابتدا عدد را به مبنای ۲ می بریم

$$(AC2)_{16} = (1010,1100,0010)_2$$

سپس تبدیل به ۸ را انجام می دهیم

$$= (101,011,000,010)_2 = (5302)_8$$

23

مکمل اعداد

• دو نوع مکمل برای هر عدد در مبنای R وجود دارد:

- مکمل R

- مکمل R-1

■ در مکمل R-1 از هر رقم مقدار (R-1) را کسر می کنیم.

■ مکمل ۹ عدد $۸۳۵_۱۰$ برابر است با $۱۶۴_۱۰$.

■ مکمل ۱۰ عدد $۱۰۱_۲$ برابر است با $۰۱۰_۲$.

■ برای یافتن مکمل R یک عدد ابتدا مکمل R-1 آن عدد را محاسبه کرده سپس مقدار آن را با آن جمع می کنیم.

■ مکمل ۱۰ عدد $۸۳۵_۱۰$ برابر است با $۱۶۴_۱۰ + ۱ = ۱۶۵_۱۰$.

■ مکمل ۲ عدد $۱۰۱_۲$ برابر است با $۰۱۰_۲ + ۱ = ۰۱۱_۲$.

24

مکمل ۲

■ تمام ارقام را نقیض کرده (0 را به 1 و 1 را به 0 تبدیل می کنیم) و با یک جمع می کنیم.

1 0 0 1 0 0 0 0

نقیض

0 1 1 0 1 1 1 1

+ 1

0 1 1 1 0 0 0 0

25

مکمل ۲

- روش سریع
1. از سمت راست شروع کنید و هر بیت را به خروجی منتقل نمایید تا زمانی که به اولین یک برسید
 2. اولین یک را نیز به خروجی منتقل کنید
 3. بقیه بیت ها را به صورت نقیض در خروجی بنویسید

1 0 0 1 0 0 0 0

0 1 1 1 0 0 0 0

نقیض

بدون تغییر

26

نکات مهم

- مکمل دو در مبنای دو معادل به قرینه کردن عدد در مبنای ده است
- جمع هر عدد با مکملش برابر با صفر خواهد بود
- در زمان محاسبه مکمل باید به بیت های صفر بی اهمیت توجه شود
- اعداد زیر با هم مساوی نیستند
 - ۱۰۱۰
 - ۰۱۰۱۰
 - ۰۰۱۰۱۰
 - ۰۰۰۱۰۱۰
- زیرا مکمل های یکسانی ندارند
- در زمان محاسبه مکمل تعداد بیت های واقعی عدد اهمیت دارد
- بیت های بی اهمیت به ۱ مبدل خواهند شد

27

مثال

- مثال: مکمل ۲ عدد ۱۰۱۰ را بدست آورید
- راه حل غلط
- ۰۱۱۰
- روش صحیح: عدد چند بیتی است؟ ۸ بیتی.
- ۰۰۰۰۱۰۱۰
- ۱۱۱۱۰۱۱۰

28

اعداد علامت دار

- در حالت طبیعی سیستم اعداد انتخاب شده برای نمایش اعداد باید بتواند هم اعداد بدون علامت و هم اعداد علامت دار را نمایش دهد.
- سه روش زیر برای نمایش اعداد علامت دار وجود دارد:
 1. نمایش بصورت اندازه-علامت.
 2. نمایش بصورت مکمل یک.
 3. نمایش بصورت مکمل دو.
- در سیستمهای کامپیتری از روش سوم استفاده می شود
- در یک عدد n بیتی علامتدار با ارزشترین بیت، بیت علامت است
- اگر بیت علامت یک باشد عدد منفی است.

29

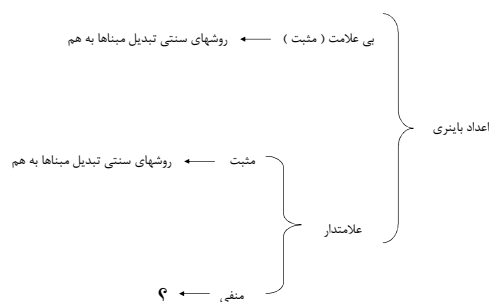
روش نمایش بصورت مکمل دو

- روی بیت علامت ۱ قرار می دهیم
- روی بقیه بیتها مکمل ۲ عدد را قرار می دهیم

$$(-1101001)_2 = \left(\begin{array}{c} \overbrace{10010111} \\ \underbrace{\quad\quad\quad}_{1101001} \end{array} \right)_2$$

30

تبدیل اعداد منفی



31

تبدیل اعداد باینری منفی به دهدهی

- هیچ روش مستقیمی برای این کار وجود ندارد
- ابتدا : قرینه عدد را تولید می کنیم
- مقدار قرینه عدد را در مبنای ده بدست می آوریم
- یک علامت منفی به جواب اضافه می کنیم
- مثال : عدد علامت دار $(10100100)_2$ را مبنای ۱۰ تبدیل کنید؟

$$\begin{array}{ccc} 10100100 & & -92 \\ \downarrow \text{مکمل ۲} & & \uparrow \\ 01011100 & \longrightarrow & 92 \end{array}$$

32

روش میانبر

-128	64	32	16	8	4	2	1
------	----	----	----	---	---	---	---

با توجه به تعداد بیت های عدد آخرین بیت را با علامت منفی در نظر می گیریم

- مثال : عدد علامت دار $(10001001)_2$ را مبنای ۱۰ تبدیل کنید؟

-128	64	32	16	8	4	2	1
1	0	0	0	1	0	0	1

$$\downarrow \qquad \qquad \qquad \downarrow \qquad \qquad \qquad \downarrow$$

$$-128 + 8 + 1 = -119$$

33

مثال : ۱۱۱۱۰۱۰۰

■ روش اول

$$\left(\begin{array}{c} 11110100 \\ -0001100 \end{array} \right)_2 = (-0001100)_2 = (-12)_{10}$$

■ روش میانبر

-128	64	32	16	8	4	2	1
1	1	1	1	0	1	0	0
↓	↓	↓	↓	↓	↓	↓	↓
-128 + 64 + 32 + 16 + 4 = -12							

34

تبدیل معکوس

- برای تبدیل یک عدد منفی به مبنای دو هیچ روش مستقیمی وجود ندارد
- ابتدا عدد را به صورت مثبت به مبنای دو تبدیل کرده ، سپس مکمل آنرا محاسبه می کنیم
- **مثال :** عدد -۸۵ را به صورت عددی ۸ بیتی و علامتدار در مبنای دو نمایش دهید.

■ ابتدا ۸۵ را به مبنای دو می بریم

128	64	32	16	8	4	2	1
↓	↓	↓	↓	↓	↓	↓	↓
0	1	0	1	0	1	0	1
=85							
↓ مکمل ۲ ↓							
1	0	1	0	1	0	1	1
= -85							

35

جمع در مبنای ۲

- جمع هر رقم می تواند یکی از حالت‌های زیر باشد
- در حاصل این جمع ها رقم سمت چپ به عنوان رقم نقلی مرحله بعد می باشد

$$\begin{array}{cccc} 0 & 0 & 1 & 1 \\ + & 0 & 1 & 0 \\ \hline 00 & 01 & 01 & 10 \end{array}$$

■ اگر رقم نقلی هر مرحله را نیز دخالت دهیم خواهیم داشت

■ رقم سمت چپ حاصل جمع به عنوان رقم نقلی مرحله بعد می باشد

رقم نقلی مرحله قبل →

$$\begin{array}{cccccccc} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ + & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ \hline 00 & 01 & 01 & 10 & 01 & 10 & 10 & 11 \\ & & & & & & & \uparrow \\ & & & & & & & 1_{36} \end{array}$$

جمع در مبنای ۲

■ مثال : حاصل جمع عدد 11110001 با عدد 00110110 در مبنای ۲ برابر است با

$$\begin{array}{r}
 11110001 \\
 + 00110110 \\
 \hline
 100100111
 \end{array}$$

37

تفریق در مبنای ۲

■ با توجه به رقم قرضی مرحله قبل می توانیم یکی از حالات زیر را داشته باشیم

■ رقم سمت چپ حاصل تفریق به عنوان رقم قرضی از مرحله بعد می باشد

رقم قرضی مرحله قبل	0	0	0	0	1	1	1	1
	0	0	1	1	0	0	1	1
	- 0	- 1	- 0	- 1	- 0	- 1	- 0	- 1
	00	11	01	00	11	10	00	11

38

تفریق در مبنای ۲

■ مثال : حاصل تفریق عدد 111000111 با عدد 10001101 در مبنای ۲ برابر است با

$$\begin{array}{r}
 111000111 \\
 - 10001101 \\
 \hline
 01011010
 \end{array}$$

39

تفریق در سیستم مکمل دو

- مکمل دو، تفریق کننده(همراه با بیت علامت) را می‌گیریم و حاصل را با تفریق شونده جمع می‌کنیم.
- اعداد باید علامت دار باشند

$$A - B = A + (-B)$$

$$\begin{array}{r} 01100111 \\ - 00001101 \\ \hline 01011010 \end{array} \xrightarrow{\text{بیت علامت}} \begin{array}{r} 01100111 \\ + 11110011 \\ \hline 101011010 \end{array} \quad \begin{array}{r} - 103 \\ + 13 \\ \hline 90 \end{array} \xrightarrow{\text{بیت علامت}} \begin{array}{r} 103 \\ - 13 \\ \hline 90 \end{array}$$

$$\begin{array}{r} 11010011 \\ - 00101000 \\ \hline 10101011 \end{array} \xrightarrow{\text{بیت علامت}} \begin{array}{r} 11010011 \\ + 11011000 \\ \hline 10101011 \end{array} \quad \begin{array}{r} - 45 \\ + 40 \\ \hline -85 \end{array} \xrightarrow{\text{بیت علامت}} \begin{array}{r} -45 \\ - 40 \\ \hline -85 \end{array}$$

40

بیت نقلی

- در تفریق اعداد علامتدار ایجاد بیت نقلی لزوماً به معنی بروز خطا نیست

- معمولاً این بیت دور ریخته میشود

- تنها حالت خطا زمانی است که نتیجه در ۸ بیت قابل ذخیره نباشد

41

توجه

- تشخیص نحوه محاسبه حاصل جمع و تفریق در مبنای ۸ و ۱۶ به عهده دانشجو می‌باشد

42

نمایش کاراکترها بوسیله کد ASCII

- **ASCII** (American Standard Code for Information Interchange)
- در کامپیوتر به حروف، ارقام، علامت ها کرکتر گفته می شود. به هر کرکتر یک کد هشت بیتی منحصر به فرد وابسته می شود که آنرا کد ASCII می نامند.

کرکتر	کد اسکی
0 تا 9	48 تا 57
A تا Z	65 تا 90
a تا z	97 تا 122

- کاراکترهای قابل چاپ

- کرکترهای قابل چاپ دارای کدهای 32 تا 126 می باشند.

کرکتر	کد اسکی
ESC	27
CR	10
LF	13

- کاراکترهای کنترلی

- کرکترهای کنترلی دارای کدهای 0 تا 31 می باشند.

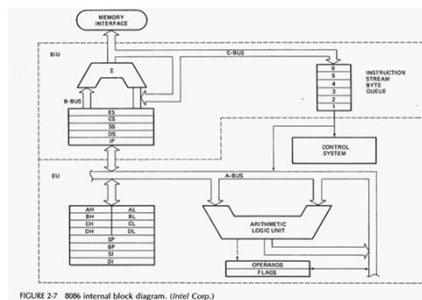
43

فصل دوم

قسمت های یک سیستم کامپیوتری

44

واحد اجرایی و واحد رابط گذرگاه

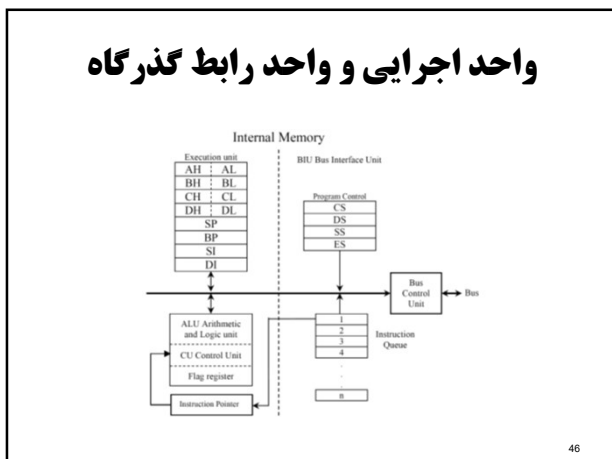


EU (Execution Unit)

BIU (Bus Interface Unit)

45

واحد اجرایی و واحد رابط گذرگاه



46

حافظه اصلی



47

- سیستم داده ها را در حافظه به ترتیب بایت معکوس ذخیره سازی می کند.
- حافظه اصلی یک مجموعه منطقی از مکان‌هایی است که هر کدام می‌تواند یک بایت دستورالعمل‌ها یا داده‌ها را ذخیره نماید. هر بایت حافظه اصلی دارای یک برچسب عددی به نام آدرس می‌باشد.
- آدرس هر بایت از حافظه اصلی را می‌توان با سگمنت حاوی بایت مزبور و به دنبال آن افستی که از ابتدای سگمنت یاد شده در نظر گرفته می‌شود، آدرس دهی کرد.

آفست OFFSET

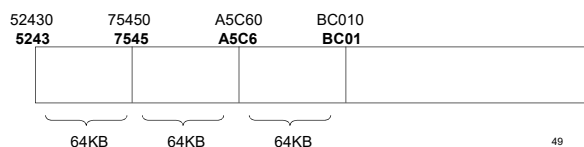


48

- فاصله از ابتدای سگمنت
- نیاز به ۱۶ بیت داریم

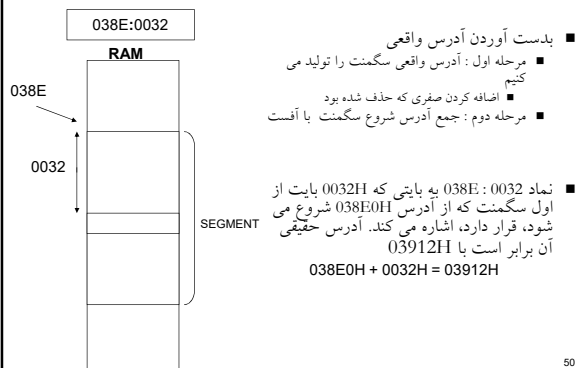
سگمنت (Segment)

- بخش هایی با اندازه 64KB
- شروع یک سگمنت باید مضرب از ۱۶ باشد
- آدرس شروع همه سگمنت ها به صفر ختم میشود .
- این صفر را حذف می کنیم.
- آدرس سگمنت ۱۶ بیتی می شود.



49

آدرسدهی SEG:OFF



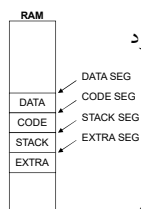
- بدست آوردن آدرس واقعی
- مرحله اول : آدرس واقعی سگمنت را تولید می کنیم
- اضافه کردن صفری که حذف شده بود
- مرحله دوم : جمع آدرس شروع سگمنت با آفست

- نماد 038E : 0032 به بیتی که 0032H بایت از اول سگمنت که از آدرس 038E0H شروع می شود، قرار دارد، اشاره می کند. آدرس حقیقی آن برابر است با 03912H

$$038E0H + 0032H = 03912H$$

50

اجزاء یک برنامه اسمبلی



- هر برنامه اسمبلی از بخشهای زیر تشکیل می شود
- Code ■
- Data ■
- Stack ■
- Extra ■
- هر یک از این بخشها باید روی RAM قرار گیرند

51

ثباتها (Registers)

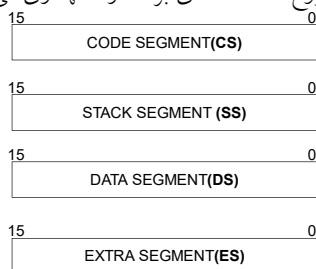
■ در روی پردازنده انواع مختلفی از ثباتها وجود دارد که عبارتند از:

- ثبات های سگمنت
- ثبات های اشاره گر
- ثبات های عمومی
- ثبات های شاخص
- ثبات پرچم

52

ثبات های سگمنت

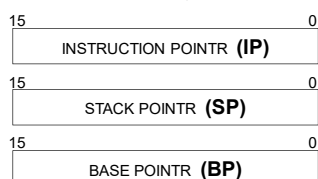
■ ثبات های ۱۶ بیتی هستند
 ■ آدرس شروع سگمنت های برنامه را نگهداری می کنند



53

ثباتهای اشاره گر

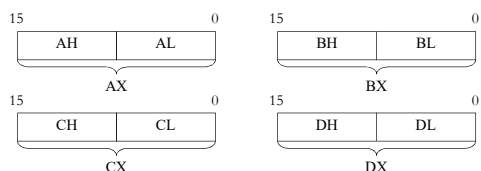
■ ثبات های ۱۶ بیتی هستند
 ■ IP: حاوی آدرس آفست دستورالعمل بعدی جهت اجرا
 ■ SP: حاوی آدرس آفست عضو بالای پشته
 ■ BP: ثبات اشاره گر پایه می باشد



54

ثبات های عمومی

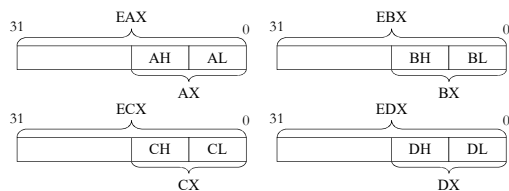
- ثبات های ۱۶ بیتی هستند
- ثبات AX: ثبات انباشت گر (Accumulator)
- ثبات BX: ثبات پایه (Base)
- ثبات CX: ثبات شمارش (Counter)
- ثبات DX: ثبات داده (Data)
- به دو بخش ۸ بیتی با ارزش (H) و کم ارزش (L) تقسیم می شوند



55

ثبات های عمومی گسترش یافته

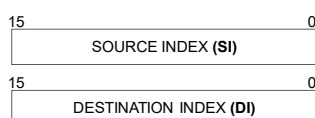
- ثبات های ۳۲ بیتی هستند
- در روی پردازشگرهای ۸۰۳۸۶ و به بعد ایجاد شده اند
- این ثباتها عبارتند از: EAX, EBX, ECX, و EDX
- ۱۶ بیت کم ارزش آنها همان ثباتهای عمومی معادل هستند.



56

ثبات های شاخص (INDEX)

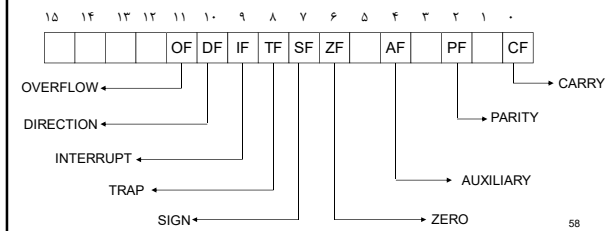
- ثباتهای ۱۶ بیتی می باشند.
- DI به عنوان ثبات مقصد شناخته می شود
- SI به عنوان ثبات شاخص مبدا شناخته می شود
- برای انتقالات بلوکی، آدرس دهی شاخص و استفاده در جمع و تفریق قابل دسترسی می باشند.



57

ثبات پرچم (FLAG)

- ثبات ۱۶ بیتی می باشد.
- برخی از دستورالعمل ها از جمله دستورات مقایسه و محاسبه وضعیت برخی بیتهای آنها تغییر می دهند.



ثبات پرچم (FLAG)

- Carry Flag: (بیت انتقال) معمولاً، از محاسبات تاثیر می پذیرد. برای اعمال جمع و تفریق از این بیت، برای عدد انتقال (رقم نقلی) استفاده می گردد. در عملی مانند Shift دادن بیتها، آخرین بیت Shift داده شده، در این بیت قرار می گیرد.
- Parity Flag: از این بیت، معمولاً برای اشکال زدایی در انتقال اطلاعات استفاده می شود. به عبارت دیگر، برای کنترل صحت اطلاعات به کار میرود.
- Auxiliary Carry Flag: از این بیت، به عنوان بیت جانشین بیت Carry استفاده می شود.
- Zero Flag: در صورتی که نتیجه عمل محاسباتی 0 شده باشد، مقدار این بیت، 1 می شود. در غیر این صورت 0 خواهد بود.

ثبات پرچم (FLAG)

- Sign Flag: به معنی علامت و برای بررسی نتیجه عملیات محاسباتی بکار می رود. یعنی اگر نتیجه عملیات منفی باشد این بیت یا ثبات برابر با ۱ وگرنه ۰ می شود.
- Trap Flag: برای اجرای دستور به دستور برنامه، از این بیت استفاده می وشد. اگر این بیت، 1 باشد، برنامه به صورت دستور به دستور اجرا می گردد.
- Interrupt Flag: با استفاده از این بیت، می توانیم کاری کنیم که وقفه ها، فعال یا غیر فعال گردند.
- IF=0 → وقفه ها غیر فعال IF=1 → وقفه ها فعال
- Direct Flag: از این بیت، معمولاً برای اعمال رشته ای استفاده می شود. جهت مقایسه یا انتقال رشته ها را نیز مشخص می کند.
- DF=1 → Right to Left DF=0 → Left to Right
- Overflow Flag: در صورتی که عمل محاسباتی دارای سرریز باشد، مقدار این بیت، 1 در غیر این صورت 0 خواهد بود.

فصل سوم

موارد ضروری برای کد نویسی در زبان اسمبلی

61

توضیحات برنامه (comment)

- با ";" آغاز می شوند.
- کلیه کاراکترها در همان سطر بعد از ";" به عنوان توضیح در نظر گرفته می شود.
- در اجرای برنامه هیچ تاثیری ندارند

62

کلمات رزرو شده

- دستورات
 - مانند ADD, MOV اعمالی هستند که کامپیوتر می تواند اجرا کند.
- پیش پردازنده ها
 - مانند SEGMENT یا END، که برای دادن اطلاعات به اسمبلر استفاده می شود.
- عملگرها
 - مانند FAR و SIZE که شما در عبارات از آن استفاده می کنید.
- سمبولهای از پیش تعیین شده
 - مانند @Data و @Model که اطلاعاتی را به برنامه شما بر می گردانند .

63

شناسه (Identifier)

- یک شناسه نامی است که به یک عنصر در برنامه داده می شود که نیاز به رجوع به آن وجود دارد.
- انواع شناسه
 - نام اشاره به آدرسی از عنصر داده دارد.
 - مانند counter در counter DB 0
 - برچسب اشاره به آدرس دستور، رویه یا سگمنت خاصی دارد.
 - مانند main در main PROC FAR

64

قانون تعریف شناسه

- می توانیم از کاراکترهای زیر استفاده کرد

کاراکترهای مجاز	نوع
z تا a و Z تا A	حروف الفبا
0 تا 9 (اولین کاراکتر نباید باشد)	ارقام
., @, \$, _ , ?	کاراکترهای خاص

- شروع شناسه باید با حروف الفبا یا کاراکترهای خاص به غیر از "." (نقطه) باشد.
- نمی توان از کلمات رزرو شده به عنوان شناسه استفاده کرد. همچنین از نام ثابت ها برای نام گذاری شناسه نمی توان استفاده کرد.

65

دستورات

- دستورات اسمبلی دارای چهار بخش می باشند که هر یک از بخشها با فاصله از هم جدا می شوند.
- نوشتن بخشهای داخل [] اختیاری می باشد

[identifier:] operation [operand(s)] [;comment]

- بخش operand(s) وابسته به بخش operation می باشد یعنی اینکه ممکن است یک دستورالعمل دو عملوند، یک عملوند داشته باشد و یا اصلا عملوند نداشته باشد .

مانند
P30: ADD AX , BX ;ADD Ax with Bx

66

پیش پردازنده PAGE و TITLE

- در شروع برنامه نوشته می شود.
- شکل کلی
PAGE [length],[width]
- حداکثر تعداد خط در یک صفحه و حداکثر ستون در یک سطر را مشخص می کند
- به عنوان مثال Page 60,132 صفحه ای با 60 سطر و 132 ستون فراهم می کند.
- اگر نوشته نشود کامپایلر مقدار پیش فرض آن یعنی Page 50,80 را در نظر خواهد گرفت.
- برای اینکه بتوان برای برنامه عنوان را مشخص نمود از پیش پردازنده title استفاده می کنیم.

67

پیش پردازنده SEGMENT

- برای تعریف سگمنتهای برنامه استفاده خواهیم کرد.
- شکل کلی آن به صورت زیر می باشد
- SEGMENT شروع سگمنت و ENDS پایان آنرا مشخص می کند.

```

name      SEGMENT      [operations]      ; begin segment
      .
      .
name      ENDS                ; end segment

```

68

برنامه EXE

- هر برنامه EXE از سگمنتهای زیر تشکیل می شود.
- STACK SEGMENT
- DATA SEGMENT
- EXTRA SEGMENT
- CODE SEGMENT
- برای تعریف این سگمنتها می توان از پیش پردازنده SEGMENT استفاده کرد.

69

قالب کلی یک برنامه EXE

```

1 Page 60,132
2 Title A04ASMI segments for an .EXE program
3 .....
4 STACKSG SEGMENT PARA STACK 'stack'
5 ..... اندازه پشته
6 STACKSG ENDS
7 .....
8 DATASEG SEGMENT PARA 'data'
9 ..... داده های برنامه
10 DATASEG ENDS
11 .....
12 CODESEG SEGMENT PARA 'code'
13 MAIN PROC FAR
14 ASSUME SS: STACKSG, DS:DATASEG, CS:CODESEG } مقدار دهی نیت‌های سگمنت
15 MOV AX, DATASEG ; set address of data
16 MOV DS, AX ; segment in DS
17 ..... کدهای برنامه
18 MOV AX, 4C00H ; End processing
19 INT 21H ; End of procedure
20 MAIN ENDP ; End of segment
21 CODESEG ENDS ; End of program
22 END MAIN

```

70

پیش پردازنده PROC

- سگمنت کد حاوی کدهای اجرایی برنامه می‌باشد.
- کدهای اجرایی در قالب یک یا بیشتر از یک رویه می‌باشند.
- تعریف هر رویه با پیش پردازنده PROC آغاز شده و با ENDP پایان می‌یابد.

NAME	OPERATION	OPERAND	COMMENT
segname	SEGMENT	PARA	
procname	PROC	FAR	; One
.	.	.	; procedure
.	.	.	; within
.	.	.	; the code
procname	ENDP	.	; segment
segname	ENDS	.	

71

یک برنامه EXE با سگمنت‌های معمولی

```

Page 60,132
Title A04ASMI segments for an .EXE program
.....
STACKSG SEGMENT PARA STACK 'stack'
DW 32 DUP(0)
STACKSG ENDS
.....
DATASEG SEGMENT PARA 'data'
FLDD DW 175
FLDE DW 150
FLDF DW ?
DATASEG ENDS
.....
CODESEG SEGMENT PARA 'code'
MAIN PROC FAR
ASSUME SS: STACKSG, DS:DATASEG, CS:CODESEG
MOV AX, DATASG ; set address of data
MOV DS, AX ; segment in DS

MOV AX, FLDD ; move 175 to AX
ADD AX, FLDE ; add 150 to AX
MOV FLDF, AX ; store sum in FLDF

MOV AX, 4C00H ; End processing
INT 21H ; End of procedure
MAIN ENDP ; End of segment
CODESEG ENDS ; End of program
END MAIN

```

72

پیش پردازنده سگمنت ساده شده

- به جای پیش پردازنده سگمنت قبلی پیش پردازنده سگمنت ساده شده استفاده کرد.
 - برای استفاده از آنها باید ابتدا مدل حافظه را مشخص کنیم. قالب آن چنین است:
- .MODEL** memory-model
- مدل های حافظه ای که می تواند استفاده شود: Small, Tiny, Compact, Medium, یا Large می باشند.
 - برای برنامه EXE عمدتاً Small استفاده می شود.
 - برای برنامه COM عمدتاً Tiny استفاده می شود.

73

پیش پردازنده سگمنت ساده شده

```

Page 60,132
Title A04ASML segments for an .EXE program
;-----
.MODEL SMALL
.STACK 64
.DATA
  FLDD DW 175
  FLDE DW 150
  FLDF DW ?
;-----
.CODE
MAIN PROC FAR
MOV AX, @data ; set address of data
MOV DS, AX ; segment in DS

MOV AX, FLDD ; move 175 to AX
ADD AX, FLDE ; add 150 to AX
MOV FLDF, AX ; store sum in FLDF

MOV AX, 4C00H ; End processing
INT 21H
MAIN ENDP ; End of procedure
END MAIN ; End of program

```

74

تعریف داده

[name] Dn expression

نوع	اندازه	توضیحات
DB	1 byte	Byte
DW	2 byte	Word
DD	4 byte	Double word
DF	16 byte	Far word
DQ	8 byte	Quad word
DT	10 byte	Ten bytes

```

Arr1 DD 15 Dup(0)
Arr2 DB 20 Dup(?)

```

- از شکل کلی زیر استفاده می شود.

- نوشتن نام اختیاری می باشد.
- Dn می تواند یکی از انواع روبرو باشد.

- تعریف آرایه با عبارت Dup انجام میشود
 - قبل از Dup تعداد
 - بعد از Dup مقدار (داخل پاراتر)
 - مثال

75

مثال تعریف رشته های کاراکتری و مقادیر عددی

```

untitled
Page 60,132
TITLE  A80DEFIN(EXE)  Define data directives
.MODEL  SMALL
.DATA
:
:      DB - Define Bytes:
:-----
0000 00          BYTE1 DB  ?          : Uninitialized
0001 30          BYTE2 DB  48          : Decimal constant
0002 30          BYTE3 DB  30h         : Hex constant
0003 7A          BYTE4 DB  01111010B   : Binary constant
0004 000A[      BYTE5 DB  10 Dup(0)      : Ten zeros
      00          ]
]

000E 50 63 20 45 60 70  BYTE6 DB  'Pc Emporium': Character string
8F 72 6B 75 6D          BYTE7 DB  '12345'      : Number as chars
0019 31 32 33 34 35          BYTE8 DB  01,'Jan', 02,'Feb',03,'Mar'
001E 01 4A 61 6E 02 46          : Table of months
65 62 03 4D 61 72          ]

:      DW - Define words
:-----
002A FF0         WORD1 DW  0FF0h         : Hex constant
002C 007A       WORD2 DW  01111010B   : Binary constant
002E 031E x     WORD3 DW  87654321h    : Address constant
0030 0002 0004 0006 0007    WORD4 DW  2,4,6,7,9      : Table of 5 constant
      0009          ]
003A 0008[      WORD5 DW  8 Dup(0)      : Six zeros
      5000          ]

:      DD - Define Doublewords
:-----
004A 00000000    DDWORD1 DD  ?          : Uninitialized
004E 0000A23A    DDWORD2 DD  41562          : Decimal value
0052 00000018 00000030    DDWORD3 DD  24 + 48          : Im constant
005A 00000001    DDWORD4 DD  BYTE3-BYTE2 :Difference
]

:      DQ - Define Quadwords
:-----
005E 0000000000000000    QWORD1 DQ  0          : Zero constant
0066 395E000000000000    QWORD2 DQ  05E39h     : Hex constant
006E 5AA7000000000000    QWORD3 DQ  41562          :Decimal constant
END

```

76

پیش پردازنده EQU

- برای تعریف ثابت ها استفاده می شود
- مثال:

Factor EQU 12

77

فصل چهارم

نوشتن برنامه های COM

78

برنامه های COM

- این برنامه های فقط به یک سگمنت محدود می باشند.
- اندازه یک برنامه COM نمی تواند بزرگتر 64k باشد.
- یک برنامه com فقط شامل سگمنت کد خواهد بود و سگمنت پشته و داده را نخواهد داشت.
- کاربر مجبور هست در صورتی که برنامه حاوی داده باشد آنها را در سگمنت کد تعریف کند.
- اولین دستور نوشته شده داخل سگمنت کد 100H org خواهد بود تا اسمبلر چینش برنامه را از آفست 100 شروع کند.
- داده های برنامه در ابتدای سگمنت کد تعریف می شوند برای این که بتوانیم از روی بخش داده ای پرش کنیم و به بخش اجرایی برنامه برسیم اولین دستور اجرایی که بعد از 100H org قرار خواهد گرفت دستور JMP می باشد.

79

نمونه مثالی از یک برنامه COM با پیش پردازنده های segment

```

PAGE 60 , 132
TITLE A07COM1 COM program to move and add
CODESEG SEGMENT PARA 'Code'
ASSUME CS:CODESEG, DS:CODESEG, SS:CODESEG, ES:CODESEG
ORG 100H ;Start at end of PSP
BEGIN: JMP A10MAIN ;Jump past data
;-----
FLDD DW 175 ;Data definitions
FLDE DW 150
FLDF DW ?
;-----
A10MAIN PROC NEAR
MOV AX , FLDD ;move 0175 to AX
ADD AX , FLDE ;add 0150 to AX
MOV FLDF , AX ;store sum in FLDF
MOV AX , 4C00H ;End processing
INT 21H
A10MAIN ENDP
CODESEG ENDS
END BEGIN

```

80

نمونه مثالی از یک برنامه COM با پیش پردازنده های سگمنت ساده شده

```

PAGE 60 , 132
TITLE A07COM2 COM program to move and add
.MODEL SMALL
.CODE
ORG 100H ;Start at end of PSP
BEGIN: JMP A10MAIN ;Jump past data
;-----
FLDD DW 175 ;Data definitions
FLDE DW 150
FLDF DW ?
;-----
A10MAIN PROC NEAR
MOV AX , FLDD ;move 0175 to AX
ADD AX , FLDE ;add 0150 to AX
MOV FLDF , AX ;store sum in FLDF
MOV AX , 4C00H ;End processing
INT 21H
A10MAIN ENDP
END BEGIN

```

81

فصل پنجم

دستورات سمبولیک و آدرس دهی

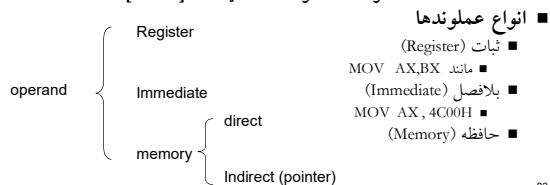
82

عملوند دستورات

- تعداد عملوند دستورات بسته به نوع دستور می تواند متفاوت باشد.
- بدون عملوند
- یک عملوند
- دو عملوند

- شکل کلی دستورات زبان اسمبلی

[label:] عملیات ۱ عملوند ۲، عملوند ۳



83

عملوند حافظه

- ارجاع مستقیم به حافظه
- به یک موقعیت مستقیم حافظه رجوع می کند.
- مانند:

```
MOV BX,WORDA
MOV BYTEA,DL
MOV CX,DS:[38B0H]
```

- ارجاع غیرمستقیم به حافظه
- برای آدرس دهی تفاوت مکانی سکمت استفاده می شود.
- ثباتهای DI, SI و BX یا ثبات DS برای پردازش داده های سکمت داده به شکل DS:SI, DS:DI و DS:BX استفاده می شوند.
- ثبات BP یا ثبات SS به شکل SS:BP برای دستکاری داده های پشته استفاده می شود.
- مانند:

```
ADD [BX],25
ADD CL,[BX]
```

84

دستور MOV

- شکل کلی دستور
- [Label:] MOV Register/Memory , Register/Memory/Immediate
- مقدار موجود در عملوند دوم خود را بر روی عملوند اول کپی خواهد کرد.
- هر دو عملوند باید اندازه یکسانی داشته باشند.
- با دستور MOV موارد زیر امکانپذیر نیست.
 - حافظه را به حافظه
 - بلافضل به ثابت Segment
 - ثابت Segment به ثابت Segment
- برای این منظور بیش از یک دستور MOV نیاز می باشد.

85

دستور انتقال و پر کردن

- اگر اندازه‌ی حافظه‌ی مبدا از مقصد کوچکتر باشد می توان از دستورات MOVZX و یا MOVZX استفاده کرد.
- شکل کلی دستور:
- [Label:] MOVZX/MOVZX Register/Memory , Register/Memory/immediate
- MOVZX برای داده‌های بدون علامت استفاده می شود .
- صفر را روی بقیه‌ی بیت‌ها کپی می کند.
- MOVSX برای داده‌های علامت دار استفاده می شود .
- بیت علامت را روی بقیه‌ی بیت‌ها کپی می کند.

86

دستور XCHG

- مقدار عملوندهای خود را جابجا می کند.
- شکل کلی
- [Label:] XCHG Register/Memory , Register/Memory

■ مثال:

XCHG CL,BH

87

دستور LEA

■ شکل کلی:

[Label:] LEA Register, Memory

■ آدرس آفست عملوند دومی بر روی عملوند اولی قرار می دهد.

■ مثال:

```
Datatl b 20 Dup(0)
...
LEA BX, Datatl b
MOV [BX], 1FH
```

88

دستور DEC و INC

■ شکل کلی

[Label:] INC/DEC Register/Memory

■ دستور INC مقدار عملوند را یک واحد افزایش می دهد.

■ دستور DEC مقدار عملوند را یک واحد کاهش می دهد.

■ در فصل بعد بصورت کامل توضیح داده خواهد شد.

89

برنامه انتقال توسعه یافته

```
Page 60,132
TITLE A06MOVE (EXE) Extended move operation
-----
.MODEL SMALL
.STACK 64
-----
.DATA
HEADG1 DB 'InterTech'
HEADG2 DB 'LaserCorp', '$'
-----
.CODE
A10MAIN PROC FAR
MOV AX, @data ; Initialize segment
MOV DS, AX ; registers
MOV ES, AX

MOV CX, 09 ; Initialize to mov 9 chars
LEA SI, HEADG1 ; Initialize address of headg
LEA DI, HEADG2 ; and HEADG2
A20:
MOV AL, [SI] ; Get character from HEADG1.
MOV [DI], AL ; move it to HEADG2
INC SI ; Incr next char in HEADG1
INC DI ; Incr nexs pos'n in HEADG2
DEC CX ; Decrement cont. for loop
JNZ A20 ; Count not zero? yes loop
; Finished
MOV AH, 09H ; Request display
LEA DX, HEADG2 ; of HEADG2
INT 21H

MOV AX, 4C00H ; End processing
INT 21H
A10MAIN ENDP
END A10MAIN
```

90

فصل ششم

دستورات محاسباتی

91

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
- ...

92

جمع و تفریق

- این عملیات ها می تواند بر اعداد علامت دار یا بدون علامت انجام شوند
- هیچ تفاوتی را روی آنها قائل نیستند.
- ADD: مقدار دو عملوند را جمع کرده و روی عملوند اول قرار می دهد.
- SUB: مقدار عملوند دوم را از عملوند اول کم کرده و روی عملوند اول قرار می دهد.
- نتیجه این عملیاتها محتوای برخی پرچمها را تغییر خواهد داد.
- شکل کلی این دستورات

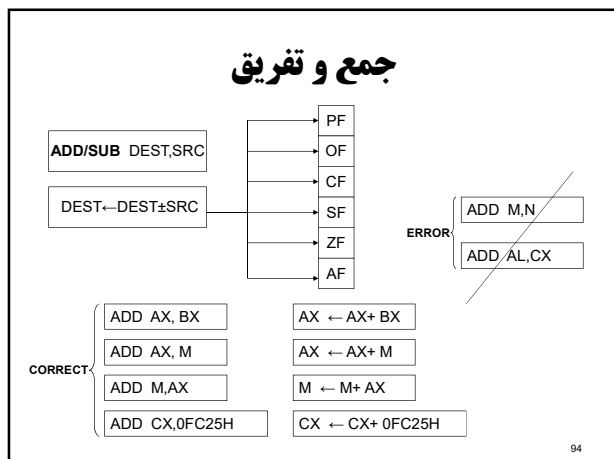
```

[label:] ADD/SUB register , register
[label:] ADD/SUB memory , register
[label:] ADD/SUB register , memory
[label:] ADD/SUB register , immediate
[label:] ADD/SUB memory , immediate

```

93

جمع و تفریق



94

جمع و تفریق

- پس از انجام محاسبه ممکن است نتیجه درست یا نادرست باشد.
- از طریق تست OF و CF می توان اعتبار محاسبات را برای اعداد علامتدار و بدون علامت سنجید.
- رقم نقلی محاسباتی
 - رقم نقلی، خروجی از با ارزشترین بیت در محاسبات دودویی می باشد که در روی پرچم CF قرار می گیرد.
 - اگر برابر صفر باشد نتیجه محاسبه برای داده های دودویی بدون علامت معتبر خواهد بود و اگر یک باشد نامعتبر.
- سرریز محاسباتی
 - سرریز محاسباتی از طریق رقم نقلی ورودی و رقم نقلی خروجی با ارزشترین بیت (سمت چپ ترین بیت) مشخص می شود.
 - اگر رقم نقلی خروجی از آخرین بیت مخالف رقم نقلی ورودی به آن باشد سرریز محاسباتی اتفاق افتاده و $OF = 1$ ، در غیر این صورت $OF = 0$ خواهد شد.
 - اگر OF برابر صفر باشد محاسبه برای داده های علامتدار صحیح بوده در غیر این صورت (یعنی OF برابر یک باشد) صحیح نخواهد بود.

95

جمع و تفریق

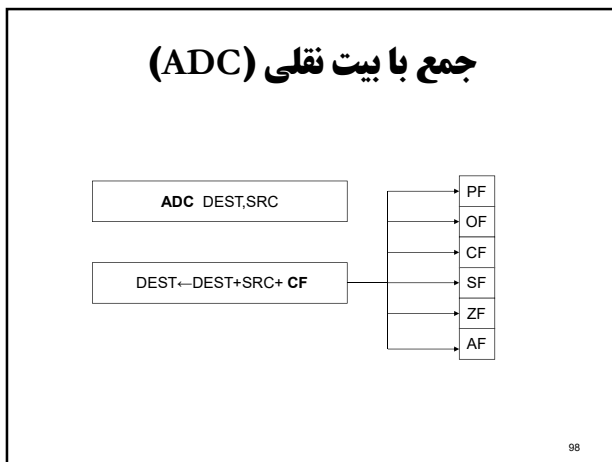
دودویی	دهدهی		CF	OF
	بدون علامت	علامتدار		
10110010 + 00011011 11001101	170 + 27 205	-28 + 27 -51	0	0
11101000 + 01110011 10110011	244 + 115 105	-12 + 115 103	1	0
01101000 + 01010111 10111111	104 + 87 191	104 + 87 -65	0	1
10010100 + 11010110 01101010	148 + 214 106	-108 + -42 +106	1	1

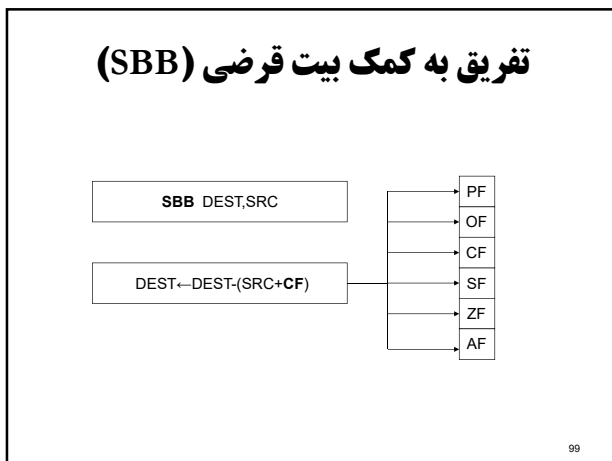
96

مثال جمع و تفریق

```

TITLE A1JADD (COM) ADD and SUB operations
.MODEL SMALL
.CODE
ORG 100H
SHORT A10MAIN
BEGIN:
BYTE1 DB 64H ;Data items
BYTE2 DB 40H
BYTE3 DB 16H
WORD1 DW 4000H
WORD2 DW 2000H
WORD3 DW 1000H
A10MAIN PROC NEAR ;Main procedure:
CALL B10ADD ;Call Add routine
CALL C10SUB ;Call SUB routine
MOV AX,4C00H ;End processing
INT 21H
A10MAIN ENDP
;
; Examples of ADD bytes:
B10ADD PROC
MOV AL,BYTE1 ;Register-to-register
MOV BL,BYTE2 ;Memory-to-register
ADD AL,BL ;Register-to-register
ADD AL,BYTE3 ;Memory-to-register
ADD BYTE1,BL ;Register-to-memory
ADD BL,10H ;Immediate-to-register
ADD BYTE1,25H ;Immediate-to-memory
RET
B10ADD ENDP
;
; Examples of SUB words:
C10SUB PROC
MOV AX,WORD1 ;Register-from-register
MOV BX,WORD2 ;Memory-from-register
SUB AX,BX ;Register-to-register
SUB AX,WORD3 ;Memory-from-register
SUB WORD1,BX ;Register-to-memory
SUB BX,1000H ;Immediate-from-register
SUB WORD1,256H ;Immediate-from-memory
RET
C10SUB ENDP
    
```





دستورالعمل های INC , DEC

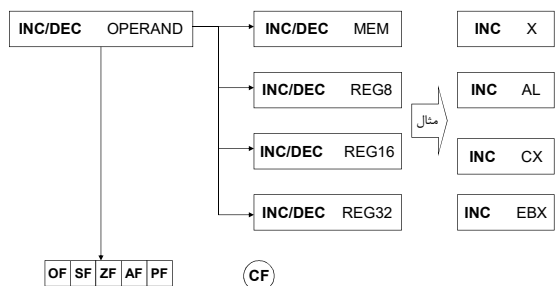
- دستورالعمل INC , DEC به ترتیب عملوند مقصد را به اندازه یک واحد کاهش و یا افزایش می دهد.
- شکل کلی این دستور بصورت زیر می باشد
[label:] INC/DEC Register/Memory

نکات:

- مقدار عملوند مقصد را به صورت یک عدد بدون علامت در نظر میگیرند.
- نشانه های OF و SF و ZF را تغییر میدهند ولی نشانه CF را تغییر نمی دهند.
- برای افزایش و کاهش شمارنده ها مفیدند و از دستورات جمع و تفریق متناظر کارآمد ترند.
- بهترین مکان برای نگه داشتن شمارنده ها در صورت امکان ثبات ها می باشند.

100

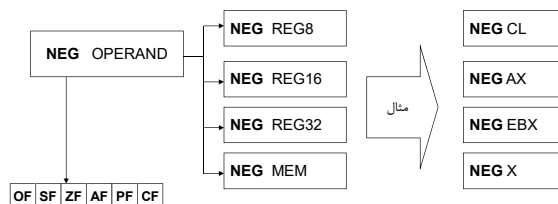
دستورالعمل های INC , DEC



101

دستورالعمل NEG

- این دستورالعمل عملوند خود را منفی می نماید یعنی مکمل ۲ آن را محاسبه می نماید.



102

دستورالعمل های ضرب

- اسمبلی دارای دو دستورالعمل ضرب می باشد :

IMUL

- عملوندها را بصورت علامتدار در نظر می گیرد.
- برای ضرب اعداد علامتدار استفاده می شود.

MUL

- عملوندها را بصورت بدون علامت در نظر می گیرد.
- برای ضرب اعداد علامتدار استفاده می شود.
- شکل کلی این دستورات بصورت زیر می باشد.

[label:] MUL/IMUL Register/Memory

103

دستورالعمل های ضرب

MUL/IMUL OPERAND8

MUL/IMUL OPERAND16

OPERAND8
 $\times \frac{AL}{AX}$

OPERAND16
 $\times \frac{AX}{DX,AX}$

MUL/IMUL OPERAND32

OPERAND32
 $\times \frac{EAX}{EDX,EAX}$

104

دستورالعمل های ضرب

- عملوند ثابت نمی تواند باشد.
- چنانچه OPR از نوع بایت باشد محتوی OPR در محتوی AL ضرب شده نتیجه در AX قرار می گیرد.
- چنانچه OPR از نوع WORD باشد محتوی OPR در محتوی AX ضرب شده نتیجه در AX : DX قرار می گیرد و محتوی ثابت های AX , DX از بین می رود.
- چنانچه OPR از نوع 32 بیتی باشد محتوی OPR در محتوی EAX ضرب شده نتیجه در EAX : EDX قرار می گیرد.
- پس از عمل ضرب ممکن است مقادیر نشانه های PF و SF و ZF تغییر کنند .

مثال:

```
MOV AL, 10
MOV X, -8
IMUL X
```

- محتوی ثابت AX برابر با 80 - می شود.

105

دستورالعمل های تقسیم

- اسمبلی دارای دو دستورالعمل تقسیم می باشد :

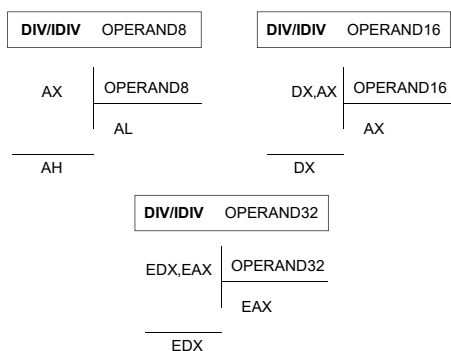
- IDIV**
 - عملوند را بصورت علامتدار در نظر می گیرد.
- DIV**
 - عملوند را بصورت بدون عملوند در نظر می گیرد.

- شکل کلی دستور

[label:] DIV/IDIV Register/Memory

106

دستورالعمل های تقسیم



107

دستورالعمل های تقسیم

- عملوند ثابت نمی تواند باشد.
- چنانچه OPR از نوع بایت باشد محتوی AX بر محتوی OPR تقسیم شده نتیجه در AL قرار می گیرد و باقیمانده تقسیم در AH قرار می گیرد.
- چنانچه OPR از نوع WORD باشد محتوی DX:AX بر محتوی OPR تقسیم شده نتیجه تقسیم در AX قرار می گیرد و باقیمانده در DX قرار می گیرد.
- چنانچه OPR از نوع 4 بایتی باشد محتوی EDX:EAX بر محتوی OPR تقسیم شده نتیجه تقسیم در EAX قرار می گیرد و باقیمانده در EDX قرار می گیرد.

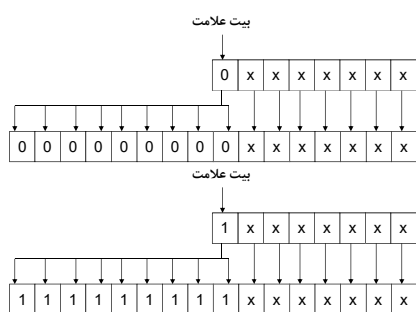
- مثال:

```
X      DB      13
MOV    AX , 134
DIV    X
```

- پس از اجرای دستورالعمل های فوق محتوی AL برابر با ۱۰ و محتوی AH برابر با ۴ می باشد.

108

گسترش BYTE به WORD



109

دستورات گسترش داده

Convert Byte to Word CBW ■

Convert Word to Doubleword CWD ■

110

فصل هفتم

موارد ضروری برنامه نویسی
برای منطق و کنترل

111

انواع آدرس‌ها

- اسمبلر از سه نوع آدرس پشتیبانی می‌کند که عبارتند از
 1. آدرس کوتاه (short)
 - محدوده‌ای در فاصله‌ی 128- تا +127 بایت می‌باشد.
 2. آدرس نزدیک (Near)
 - محدوده‌ای در فاصله‌ی 32768- تا +32767 بایت داخل همان سگمنت را مشخص می‌کند.
 3. آدرس دور (Far)
 - فاصله‌ی بیشتر از 32k را مشخص می‌کند.
- برخی از دستورات مانند LOOP، CALL، JMP و JNN می‌توانند همه یا برخی از این آدرس‌ها را پشتیبانی کنند. به عنوان مثال دستور JMP همه‌ی این آدرس‌ها را پشتیبانی می‌کند.

112

دستور JMP

- دستور JMP شبیه goto در می‌باشد.
 - می‌توان به بخش مشخص شده‌ای از برنامه بدون هیچ شرطی پرش نمود.
 - این دستور دارای شکل کلی زیر است:
- [label:] JMP short, near, or far address

113

مثال استفاده از دستور JMP

```

Page 60 , 132
TITLE A08JUMP (COM) using JMP for looping
.MODEL SMALL
.CODE
0100          ORG 100H
0100          A10MAIN PROC NEAR
0100  B8 0001  MOV AX , 01
0103  BB 0001  MOV BX , 01
0106  B9 0001  MOV CX , 01
0109          A20:
0109  05 0001  ADD AX , 01      ;Add 01 to AX
010C  03 D8   ADD BX , AX      ;Add AX to BX
010E  D1 E1   SHL CX , 1      ;Double CX
0110  EB F7   JMP A20      ;Jump to A20 label
0112          A10MAIN ENDP
          END A10MAIN

```

114

دستور LOOP

- برای ایجاد حلقه‌ها می‌توان استفاده کرد.
- تعداد تکرارهای حلقه را در روی ثبات CX قرار دهیم.
- هر بار که دستور LOOP اجرا می‌شود مقدار CX را یک واحد کاهش می‌دهد و در صورتی که CX مخالف صفر باشد به بر حسب مشخص شده پرش خواهد کرد.
- این دستور فقط از آدرس‌های کوتاه پشتیبانی می‌کند یعنی پرش می‌تواند از در فاصله‌ی 128- تا 127+بایت باشد.
- شکل کلی دستور LOOP

[label:] LOOP short address

115

مثال استفاده از دستور LOOP

```

Page 60 , 132
TITLE A08LOOP (COM) Illustration of LOOP
.MODEL SMALL
.CODE
0100          ORG 100H
0100          A10MAIN PROC NEAR
0100 B8 0001   MOV AX , 01      ; Initialize AX ,
0103 B8 0001   MOV BX , 01      ; BX , and
0106 BA 0001   MOV DX , 01      ; DX to 01
0109 B9 000A   MOV CX , 10      ; Initialize for
010C          A20:              ; ten loops
010C 40        INC AX           ; Add 01 to AX
010D 03 D8     ADD BX , AX      ; Add AX to BX
010F D1 E2     SHL DX , 1      ; DOUBLE DX
0111 E2 F9     LOOP A20        ; Decrement CX ,
                                ; LOOP if nonzero
0113 B8 4C00   MOV AX , 4C00H   ; End processing
0116 CD 21     INT 21H
0118          A10MAIN ENDP
END A10MAIN

```

116

دستور CMP

- این دستور (Compare) جهت مقایسه‌ی دو فیلد داده‌ای استفاده می‌شود.
- شکل کلی دستور
[label:] CMP register/memory, register/memory/immediate
- پس از اجرای این دستور پرچم‌های OF, SF, ZF, AF, PF, CF تغییر پیدا خواهند کرد.
- دستورالعمل CMP مانند دستورالعمل SUB عمل نموده ولی نتایج در جایی ذخیره نمی‌شود بلکه محتوی فلگ‌ها را تغییر می‌دهد.
- از طریق مقدار این پرچم‌ها دستورات بعدی می‌توانند تصمیم خاصی را اتخاذ کنند.

مثال ■

```

CMP AX , BX
JE A20
...
A20:

```

پرش کن اگر برابر باشد

117

پرش شرطی

- پرشهای شرطی به برنامه نویس این امکان را می دهد که ساختارهای IF و سایر ساختارهای کنترلی را ایجاد نماید.
 - شکل کلی بصورت زیر می باشد:
- [label:] Jnnn short address
- Jnnn تعیین کننده وضعیتی است که تحت آن ، پرش اجرا می شود. اگر شرط تحقق یابد، پرش صورت خواهد گرفت، در غیر این صورت دستورالعمل بعدی اجرا خواهد گردید.
 - دستورات پرش شرطی بر اساس وضعیت برخی از پرچم ها تصمیم می گیرند که به یک پرچسب پرش کنند یا نه.
 - این دستورات فقط می تواند به آدرس های کوتاه پرش کننده.

118

دستورات پرش شرطی

- پرشهای مربوط به داده های بدون علامت (منطقی)

پرچم های تست شده	توضیح	سمبول
ZF	پرش مساوی یا پرش صفر	JE/JZ
ZF	پرش غیرمساوی یا پرش غیر صفر	JNE/JNZ
CF,ZF	پرش بیشتر یا پرش کمتر نبودن یا مساوی	JG/JNBE
CF	پرش بیشتر نبودن یا مساوی یا پرش کمتر نبودن	JAE/JNB
CF	پرش کمتر یا پرش بیشتر نبودن یا مساوی	JB/JNAE
ZF,CF	پرش کمتر یا مساوی یا بیشتر نبودن	JBE/JNA

119

دستورات پرش شرطی

- پرشهای مربوط به داده های علامتدار (محاسباتی)

پرچم های تست شده	توضیح	سمبول
ZF	پرش مساوی یا پرش صفر	JE/JZ
ZF	پرش غیرمساوی یا پرش غیر صفر	JNE/JNZ
OF,SF,ZF	پرش بزرگتر یا پرش غیر کوچکتر یا مساوی	JA/JNLE
OF,SF	پرش غیر بزرگتر یا پرش غیر کوچکتر	JGE/JNL
OF,SF	پرش کوچکتر یا پرش غیر بزرگتر یا مساوی	JL/JNGE
OF,SF,ZF	پرش کوچکتر یا مساوی یا پرش غیر بزرگتر	JLE/JNG

120

دستورات پرش شرطی

■ پرشهای محاسباتی خاص

برچم های تست شده	توضیح	سمبول
هیچ یک	پرش اگر CX صفر است	JCXZ
CF	پرش اگر رقم نقلی وجود دارد	JC
CF	پرش اگر رقم نقلی وجود ندارد	JNC
OF	پرش اگر سرریز وجود دارد	JO
OF	پرش اگر سرریز وجود ندارد	JNO
PF	پرش اگر توازن وجود دارد یا توازن زوج است	JP/JPE
PF	پرش اگر توازن وجود ندارد یا توازن فرد است	JNP/JPO
SF	پرش اگر علامت دارد (منفی)	JS
SF	پرش اگر علامت ندارد (مثبت)	JNS

121

دستور CALL و دستور RET

- عبارت PROC تعریف کننده Procedure (رویه) می باشد .
- هر رویه با عبارت PROC آغاز شده و با عبارت ENDP خاتمه پیدا می کند.
- برای فراخوانی رویه می توان از دستور CALL استفاده کرد و برای برگشتن از رویه به رویه فراخوانی کننده می توان از دستور RET استفاده کرد.
- زمانی که با استفاده از دستور CALL یک رویه فراخوانی می شود، اجرای برنامه به اولین سطر رویه مشخص شده منتقل می شود، دستورات رویه سطر به سطر اجرا شده و نهایتاً با اجرای دستور RET کار رویه تمام شده و اجرای برنامه از دستور بعد از CALL ادامه پیدا می کند.

122

دستور CALL و دستور RET

- داخل یک رویه RET آخرین دستوری است که اجرا خواهد شد و کل دستورات بعد از آن داخل همان رویه اجرا نخواهد شد.
- یک روال می تواند FAR و یا NEAR باشد.
- یک روال NEAR در همان سگمنت کدی که فراخوانی می شود تعریف می گردد و یک روال FAR معمولاً در یک سگمنت کدی مجزائی تعریف می شود.
- برای اطلاعات بیشتر راجع به پیش پردازنده PROC به اسلاید ۷۱ مراجعه کنید.

123

انتقال مقادیر به یک روال و یا بلعکس

- بطرق مختلفی می توان مقادیر را به روال هایی به زبان های اسمبلی یا بلعکس انتقال داد.
- دو روش ممکن برای انتقال یک مقدار به اندازه WORD عبارتند از:
 - قرار دادن مقدار مورد نظر در یک ثابت
 - قرار دادن مقدار مورد نظر روی پشته

124

دستور CALL و دستور RET

```

page 60,132
TITLE A08CALLP (EXE) Calling procedure
.MODEL SMALL
.STACK 64
.DATA
;-----
.CODE
A10MAIN PROC FAR
0000      CALL     B10           ; call B10
0000 E8 0008 r
;
0003 B8 4C00      MOV     AX, 4C00H           ; End processing
0006 CD 21        INT     21H
0008 A10MAIN ENDP
;-----
B10 PROC NEAR
0008      CALL     C10           ; call C10
0008 E8 000C r
;
0008 C3          RET             ; return to
000C B10 ENDP           ; caller
;-----
C10 PROC NEAR
000C      RET             ; Return to
000C C3          RET             ; caller
000D C10 ENDP
;-----
END A10MAIN

```

125

فصل هشتم

عملیات دودویی

126

دستورات بول

- شکل کلی دستورات بولی
[label] operator register/memory, register/memory/immediate
- اجرای این دستورات پرچم های CF، OF، DF، SF، ZF را تغییر خواهد داد. دستورات بول که در زبان اسمبلی وجود دارند عبارتند
- AND حاصل بیت های متناظر را یک قرار خواهد داد اگر هر دو بیت متناظر برابر یک باشد.
- OR بیت متناظر را یک قرار خواهد داد اگر حداقل یکی از بیت ها برابر یک باشد.
- XOR نتیجه ی صفر خواهد داشت، اگر بیت های متناظر برابر باشند، و نتیجه ی یک خواهد داشت اگر بیت های متناظر مخالف هم باشند.
- پرچم ها را مانند عملیات AND تنظیم خواهد کرد با این تفاوت که مقدار عملوند اول تغییر نخواهد کرد.
- NOT فقط یک عملوند خواهد داشت از نوع ثنائی یا حافظه این دستور بر روی عملوند خود کلیه بیت های صفر را به یک و یک را به صفر تغییر خواهد داد.

127

```

Page 60 , 132
TITLE A08CASE (COM) Change uppercase to lowercase
.MODEL SMALL
.CODE
ORG 100H
BEGIN: JMP A10MAIN
;-----
CONAME DB 'INTERTECH SYSTEM', '$'
;-----
A10MAIN PROC NEAR
LEA BX, CONAME+1
MOV CX, 15
A20:
MOV AH, [BX]
CMP AH, 41H
JB A30
CMP AH, 5AH
JA A30
XOR AH, 00100000B
MOV [BX], AH
A30:
INC BX
LOOP A20
MOV AH, 09H
LEA DX, CONAME
INT 21H
MOV AX, 4C00H
INT 21H
A10MAIN ENDP
END BEGIN

```

مثال: تبدیل حروف بزرگ به کوچک

128

شیفت بیتها

- دستورالعملهای شیفت، بیت های واقع در موقعیت داده شده بوسیله عملوند مقصد را بطرف چپ یا راست حرکت می دهند.
- شکل کلی دستورات شیفت:
[label] shift register/memory, CL/immediate
- در پردازنده های 8088\8086 فقط شیفت یک واحدی پذیرفته شده می باشد.
- عملوند دوم فقط باید عدد یک باشد.
- اگر تعداد شیفت ها بیش از یک باشد باید ابتدا تعداد شیفتها را داخل ثبات CL قرار دهیم و این ثبات را در عملوند دوم دستور شیفت استفاده کنیم در پردازنده های رده بالاتر این کار لزومی ندارد.

129

شیفت به راست بیتها

■ برای این منظور دو دستور وجود دارد.

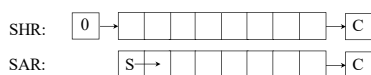
SHR ■

■ برای شیفت منطقی به راست (برای داده های بدون علامت)

SAR ■

■ برای شیفت محاسباتی به راست (برای داده های علامت دار)

■ یک واحد شیفت به راست باعث می شود عدد تقسیم بر ۲ شود.



130

شیفت به چپ بیتها

■ برای این منظور دو دستور وجود دارد.

SHL ■

■ برای شیفت منطقی به چپ (برای داده های بدون علامت)

SAL ■

■ برای شیفت محاسباتی به چپ (برای داده های علامت دار)

■ یک واحد شیفت به چپ باعث می شود عدد ضرب در ۲ شود.



131

چرخش بیت ها به راست

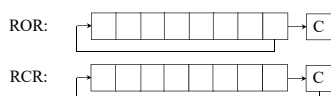
■ برای این منظور دو دستور وجود دارد.

ROR ■

■ برای چرخش منطقی به راست برای داده های بدون علامت استفاده می شود

RCR ■

■ برای چرخش با رقم نقلی به راست برای داده های علامت دار استفاده می شود.



132

چرخش بیت ها به چپ

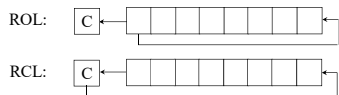
■ برای این منظور دو دستور وجود دارد.

ROL ■

■ برای چرخش منطقی به چپ برای داده های بدون علامت استفاده می شود

RCL ■

■ برای چرخش با رقم نقلی به چپ برای داده های علامت دار استفاده می شود.



133

مثال: استفاده از جدول پرش

```

TITLE A08JMPTB (EXE) Using a jump table
.MODEL SMALL
.STACK 64
.DATA
CUSTTAB DW B10CDE0
         DW B11CDE1
         DW B12CDE2
         DW B13CDE3
         DW B14CDE4
000A 43 6F 64 65 20 30 MESSG0 DB 'Code 0 processing' , '$'
      20 70 72 6F 63 65
      73 73 69 6E 67 24
001C 43 6F 64 65 20 31 MESSG1 DB 'Code 1 processing' , '$'
      20 70 72 6F 63 65
      73 73 69 6E 67 24
002E 43 6F 64 65 20 32 MESSG2 DB 'Code 2 processing' , '$'
      20 70 72 6F 63 65
      73 73 69 6E 67 24
0040 43 6F 64 65 20 33 MESSG3 DB 'Code 3 processing' , '$'
      20 70 72 6F 63 65
      73 73 69 6E 67 24
0052 43 6F 64 65 20 34 MESSG4 DB 'Code 4 processing' , '$'
      20 70 72 6F 63 65
      73 73 69 6E 67 24
;-----

```

134

مثال: استفاده از جدول پرش

```

.CODE
.386
0000 A10MAIN PROC FAR
0000 B8 ---- R      MOV AX,@data ; initialize
0003 8E D8          MOV DS, AX ; segment
0005 8E C0          MOV ES, AX ; registers
0007 E8 000F R     CALL B10JUMP
000A B8 4C00        MOV AX, 4C00H ; End processing
000D CD 21          INT 21H
000F A10MAIN ENDP
;
000F B10JUMP PROC NEAR
000F B4 10          MOV AH, 10H ; Get KB char
0011 CD 16          INT 16H ; into AL
0013 24 07          AND AL, 00001111B ; clear left 5 bits
0015 0F B6 D8       MOVZX BX, AL ; move AL to BX
0018 D1 E3          SHL BX, 01 ; Double value
001A FF A7 0000 R   JMP [CUSTTAB+BX] ; jump to cust_rtn
001E 8D 16 000A R   B10CDE0: LEA DX, MESSG0 ; code 0 routine
0022 EB 1D 90        JMP B90
0025 8D 16 001C R   B11CDE1: LEA DX, MESSG1 ; code 1 routine
0029 EB 16 90        JMP B90
002C 8D 16 002E R   B12CDE2: LEA DX, MESSG2 ; code 2 routine
0030 EB 0F 90        JMP B90
0033 8D 16 0040 R   B13CDE3: LEA DX, MESSG3 ; code 3 routine
0037 EB 08 90        JMP B90
003A 8D 16 0052 R   B14CDE4: LEA DX, MESSG4 ; code 4 routine
003E EB 01 90        JMP B90
0041 B90 :
0041 B4 09          MOV AH, 09H ; Display
0043 CD 21          INT 21H
0045 C3            RET
0046 B10JUMP ENDP
A10MAIN

```

135

فصل نهم

مقدمه‌ای بر پردازش صفحه کلید و صفحه نمایش

136

صفحه نمایش

- صفحه‌ی نمایش در حالت متنی از ۲۵ سطر و ۸۰ ستون تشکیل شده است.
- مختصات گوشه‌ی سمت چپ بالای صفحه برابر (0,0) و گوشه‌ی سمت راست پایین صفحه برابر (24,79) (در مبنای ۱۶، 18,4F) می‌باشد.

137

صفحه نمایش

- انتقال مکان نما
- می‌توان از تابع 02H وقفه‌ی شماره 10H استفاده کرد.
 - شماره صفحه را در ثبات BH (معمولاً برابر صفر است)
 - شماره سطر را در ثبات DH
 - شماره ستون را در ثبات DL قرار دهیم.
 - مثال: مکان نما به سطر هشتم (08H) و ستون پانزدهم (0FH) از صفحه‌ی شماره صفر منتقل خواهد شد.

MOV AH, 02H	: درخواست تنظیم مکان نما
MOV BH, 00	: شماره صفحه صفر
MOV DH, 08H	: سطر ۸
MOV DL, 0FH	: ستون ۱۵
INT 10H	: فراخوانی سرویس وقفه

138

صفحه نمایش

پاک کردن صفحه نمایش

- می توان از تابع 06H از وقته ی 10H استفاده کرد. باید ثابتهای که در زیر آمده اند به ترتیب مقادیردهی شوند.
 - AH = تابع 06H.
 - AL = شماره خطهایی که حرکت طوماری دارند، صفر برای کل صفحه.
 - BH = مقدار صفت (رنگ قلم، رنگ پس زمینه و حالت چشمک زن)
 - CX = ستون : سطر شروع
 - DX = ستون : سطر پایان.
- خصوصیات صفحه مشخص کننده رنگ قلم و رنگ پس زمینه می باشد که یک عدد یک بیتی می باشد. بیت های صفر تا سه از این عدد مشخص کننده ی رنگ قلم بیت های 4 تا 6 مشخص کننده رنگ پس زمینه و بیت هفتم مشخص کننده حالت چشمک زن می باشد.

MOV AX,0600H	AH=0 (حرکت طوماری) ;
MOV BH,71H	پس زمینه سفید (7) ، پیش زمینه آبی (1) ;
MOV CX,0000H	ستون : سطر بالای چاپ ;
MOV DX,184FH	ستون : سطر پایین راست ;
INT 10H	فراخوانی سرویس وقته ;

139

صفحه نمایش

چاپ رشته در خروجی

- از تابع شماره 09H وقته ی 21H می توان استفاده کرد.
- این تابع کل کاراکترهای رشته از اولین کاراکتر تا رسیدن به کاراکتر \$ را در خروجی چاپ می کند.
- برای این منظور باید 09H را روی ثبات AH و آدرس آگست اولین خانه ی رشته را روی DX قرار دهیم. مثال زیر نمونه ای از کاربرد این تابع را نشان می دهد.

```

Custmsg DB 'Hello','$'
...
Mov AH,09H
LEA DX,Custmsg
INT 21H

```

- با فراخوانی وقته 21H رشته Custmsg که برابر Hello می باشد در خروجی چاپ خواهد شد.

140

صفحه نمایش

مثال: نمایش مجموعه کاراکترهای ASCII

TITLE	page 66,132	(COM) Display ASCII character set
ADDRESS	MODEL	SMALL
CODE	ORG	100H
BEGIN:	JMP	SHORT A10MAIN
ASCHAR	DB	99,'\$'
		;Display character
		Main procedure:
A10MAIN	PROC	NEAR
	CALL	B10SCRN
	CALL	C10CURS
	CALL	D10DISP
	MOV	AX,4C0H
	INT	21H
	ENDP	
A10MAIN	END	
		Display ASCII characters:
B10SCRN	PROC	
	MOV	CX,256
	MOV	DX,ASCHAR
	INT	21H
	REF	
	ENDP	
B10SCRN	END	
		;Initialize 256 attentions
		;Initialize address of ASCHAR
C10CURS	PROC	
	MOV	AH,09H
	INT	21H
	INC	ASCHAR
	LOOP	D20
	RET	
	ENDP	
C10CURS	END	
		;Display ASCII ASCHAR
		;Increment for next character
		;Decrement CX, loop nonzero
		;Return to caller
D10DISP	PROC	
	MOV	CX,99
	LEA	DX,ASCHAR
	INT	21H
	INC	ASCHAR
	LOOP	D20
	RET	
	ENDP	
D10DISP	END	
		;Initialize 256 attentions
		;Initialize address of ASCHAR
D20:	MOV	AH,09H
	INT	21H
	INC	ASCHAR
	LOOP	D20
	RET	
	ENDP	
D20:	MOV	AH,09H
	INT	21H
	INC	ASCHAR
	LOOP	D20
	RET	
	ENDP	
D10DISP	END	
		;Display ASCII ASCHAR
		;Increment for next character
		;Decrement CX, loop nonzero
		;Return to caller
BEGIN		

141

صفحه کلید

تابع OAH از وقفه 21H برای ورودی صفحه کلید
 باید ابتدا در بخش داده‌ای لیست پارامترهای مورد نیاز آن را مشخص کنیم. مثال:

Paralist	Label	Byte
Maxlen	DB	20
Actlen	DB	?
KBData	DB	20 Dup(' ')

■ هنگام فراخوانی این وقفه در ثبات AH مقدار 0AH و آدرس آفست لیست پارامترها را در روی DX قرار خواهیم داد. مثال:

```
Mov AH , 0AH
LEA DX , Paralist
INT 21H
```

142

■ مثال: برنامه پذیرش و نمایش نام

```

PAGE 60,132
TITLE نام
;-----
A2: E10CENT PROC NEAR
;-----
PROMPT DB 'Name?', '$'
;-----
SHR DL,1 ; divide length by 2
NEG DL ; revers sign
ADD DL,40 ; add 40
MOV DH,12 ; center row
CALL Q20CURS ; set cursor
MOV AH,09H
LEA DX,KBNAME ; display name
INT 21H
RET
E10CENT ENDP
;-----
A10MAIN PROC
MOV DX,0000 ; set cursor to 00,00
CALL Q20CURS ; Display prompt
CALL C10INPT ; provide for input of name
CALL Q10CLR ; clear screen
CMP ACTLEN,00 ; name entered?
JE A30 ; no , exit
CALL D10CODE ; set bell and $
CALL E10CENT ; center , display name
JMP A20LOOP
A30:
MOV AX,4C00H ; End processing
INT 21H
A10MAIN ENDP

```

143

فصل دهم

نکات پیشرفته پردازش صفحه کلید

144

صفحه کلید

- صفحه کلید سه نوع اصلی کلید دارد
- کلیدهای استاندارد، شامل حروف A تا Z، شماره های ۰ تا ۹ و برخی کاراکترها مانند %، \$، # و ...
- کلیدهای تابعی توسعه یافته که شامل:
 - کلیدهای تابعی برنامه مانند <F1> و <F1>+<Shift> .
 - کلیدهای عددی با حالت Num Lock خاموش: <End>، <Home>، <Arrows>، ، <Ins>، <Page Up> و <Page Down> و کلیدهایی که در صفحه کلید مانند همین کلیدها عمل می‌کنند.
 - <Alt> + حروف الفبا و کلیدهای تابعی برنامه + <Alt>
- کلیدهای کنترلی <Alt>، <Ctrl> و <Shift> که در ارتباط با سایر کلیدها عمل می‌کنند.

145

وضعیت شیفت صفحه کلید

- در موقعیت 40:17H یک بایت قرار گرفته که این بایت اولین بایت وضعیت شیفت صفحه کلید می‌باشد. هریک از بیت‌های این بایت وضعیت کلیدهایی را که در جدول زیر نشان داده شده مشخص خواهند کرد.

Bit	Action	Bit	Action
7	Insert active	3	<Alt> pressed
6	CapsLock site active	2	<Ctrl> pressed
5	NumLock site active	1	<Left Shift> pressed
4	Scroll Lock site active	0	<Right Shift> pressed

- بایت دوم از وضعیت صفحه کلید برای صفحه کلید توسعه یافته در 40:18H قرار دارد.

146

وضعیت شیفت صفحه کلید (ادامه)

- بایت دیگری 40:96H قرار دارد که وضعیت برخی دیگر از کلیدهای صفحه را نشان می‌دهد. عناصری که بر ما اهمیت دارد بیت ۴ است که وقتی ۱ باشد یعنی یک صفحه کلید توسعه یافته نصب شده است.

Bit	Action	Bit	Action
7	Insert pressed	3	Ctrl/Numlock (pause) active
6	Caps Lock pressed	2	Sys Reg pressed
5	Num Lock pressed	1	Left Alt pressed
4	Scroll Look pressed	0	Left ctrl pressed

147

بافر صفحه کلید

■ یک عنصر سودمند در ناحیه داده BIOS در 40:1EH به نام بافر صفحه کلید است.

■ بافر صفحه کلید کدهای تولید شده توسط صفحه کلید را به ترتیب نگهداری می‌کند تا اینکه وقفه‌ای (مانند وقفه 16h) از آن کلیدی را تقاضا کند. سپس بافر به ترتیب این کدها را برگشت می‌دهد.

148

ورودی صفحه کلید با انعکاس

تابع 01H از وقفه 21H

- اگر در بافر صفحه کلید، کدی وجود داشته باشد آنرا برگشت می‌دهد و گرنه منتظر فشار دادن کلید از کاربر می‌ماند.
- کد کلید فشار داده شده روی AL برگشت می‌دهد.
- این تابع توانایی خواندن کلیدهای تابعی توسعه یافته (کلیدهای دو کده) را دارا می‌باشد.

MOV AH, 01H	: درخواست ورودی صفحه کلید
INT 21H	: فراخوانی سرویس وقفه
CMP AL, 00	: آیا کلید تابعی توسعه یافته فشار داده شده؟
JNZ ...	: نه کاراکتر: ASCII
INT 21H	: به تکرار عملیات برای خواندن کد دوم
...	: برای کد پیمایش

149

I/O کنسول مستقیم

تابع 06H از وقفه 21H

- برای خواندن کلید از صفحه کلید استفاده می‌شود.
- تقریباً شبیه تابع 01H می‌باشد با این تفاوت که پس از تست کردن بافر صفحه کلید اگر کلیدی وجود نداشته باشد دیگر منتظر فشار دادن کلید نمی‌ماند.
- اگر در بافر کلیدی نباشد: این تابع پرچم صفر (ZF) را برابر یک قرار داده و برای ورودی منتظر نمی‌ماند.
- اگر در بافر کلیدی باشد: این عملیات کد را در ثبات AL قرار داده و پرچم صفر را برابر صفر قرار می‌دهد.
- این عملیات کاراکتر را بر روی صفحه نمایش منعکس نمی‌کند و همچنین کلیدهای <Ctrl>+<prtsc> یا <Ctrl>+<Break> را بررسی نمی‌کند.

150

ورودی مستقیم صفحه کلید بدون انعکاس

تابع 07H از وقفه 21H

■ این تابع شبیه تابع 01H عمل می‌کند با این تفاوت که کاراکتر را بر روی صفحه منعکس نکرده و کلید <Break> + <Ctrl> واکنش نشان نمی‌دهد.

تابع 08H از وقفه 21H

■ این تابع شبیه 01H عمل می‌کند با این تفاوت که کلید فشار داده شده را بر روی صفحه نمایش منعکس نمی‌کند.

151

ورودی صفحه کلید بافر شده

تابع 0AH از وقفه 21H

■ این تابع در مباحث قبلی توضیح داده شده است.

152

بررسی وضعیت صفحه کلید

تابع 0BH از وقفه 21H

■ این تابع بافر صفحه کلید را بررسی کرده و اگر در بافر کلیدی در دسترس باشد در روی AL عدد FFH را قرار می‌دهد و در غیر اینصورت (هیچ کلیدی در دسترس نباشد) 00H را قرار خواهد داد.

153

پاک کردن بافر صفحه کلید و برداشتن تابع

- تابع 0CH از وقفه 21H
- این تابع در ارتباط با توابع 01H ، 06H ، 08H یا 0AH استفاده می‌شود .
 - تابع مورد نیاز در AL قرار می‌گیرد

MOV AH, 0CH	: درخواست ورودی صفحه کلید
MOV AL, FUNCTION	: تابع مورد نیاز
MOV DX, ABAREA	: ناحیه ورودی صفحه کلید
INT 21H	: فراخوانی سرویس وقفه

154

خواندن یک کاراکتر

تابع 00H از وقفه 16H

- این تابع فقط کلیدهای صفحه کلید ۸۳ تایی را دستکاری می‌کند و ورودی از کلیدهای اضافی بر روی صفحه کلیدهای گسترش یافته مانند <F11> و <F12> نمی‌پذیرد.

155

بازگرداندن وضعیت شیفت جاری

تابع 02H از وقفه 16H

- این تابع وضعیت شیفت صفحه کلید را که در ناحیه داده‌ای BIOS در موقعیت 40:17H قرار دارد در روی AL برگشت می‌دهد .

156

نوشتن بر روی صفحه کلید

تابع 05H از وقفه 16H

- این تابع به ما این اجازه را می‌دهد که بتوانیم کلیدی را در بافر صفحه کلید بنویسیم. همانند اینکه کلیدی فشار داده شده است.
- کاراکتر ASCII را در CH و کد پیمایش آنرا در CL وارد می‌کنیم.
- تا زمانی که بافر صفحه کلید پر شود می‌توان این عملیات را تکرار کرد.

157

```

TITLE A14ASB (COM) Convert ASCII to binary format
.MODEL SMALL
.CODE
.ORG 100H
BEGIN: JMP SHORT A10MAIN
; -----
ASCVAL DB '1234' ;Data items
BINVAL DW 0
ASCLEN DW 4
MULFACT DW 1
; -----
A10MAIN PROC NEAR ;Main procedure
CALL B10CONV
MOV AX,4C00H
INT 21H ;End processing
ENDP
B10CONV PROC NEAR ;Mult factor
MOV BX,10 ;Count for loop
MOV CX,04 ;Address of ASCVAL
LEA SI,ASCVAL+3
B20: MOV AL,[SI] ;Select ASCII character
AND AX,000FH ;Remove 3-zone
MUL MULFACT ;Multiply by 10 factor
ADD BINVAL,AX ;Add to binary
MOV AX,MULFACT ;Calculate next
MUL BX ; 10 factor
MOV MULFACT,AX ;Last ASCII character?
DEC SI ; no, continue
LOOP B20 ; yes, return
B10CONV ENDP
END BEGIN

```

شکل ۵-۱۴ تبدیل اعداد ASCII به قالب دودویی

158

```

TITLE A14BINAS (COM) Convert binary data to ASCII
.MODEL SMALL
.CODE
.ORG 100H
BEGIN: JMP SHORT A10MAIN
; -----
ASCVAL DB 4 DUP(' ','$') ;Data items
BINVAL DW 04D2H
; -----
A10MAIN PROC NEAR ;Main procedure
CALL B10CONV
MOV AH,09H ;Display
LEA DX,ASCVAL ; ASCII value
INT 21H
MOV AX,4C00H ;End processing
INT 21H
ENDP
B10CONV PROC NEAR ;Division factor
MOV CX,0510 ;Address of ASCVAL
LEA SI,ASCVAL+3 ;Get binary amount
MOV AX,BINVAL
B20: CMP AX,CX ;Value < 10?
JB B30 ; yes, exit
XOR DX,DX ;Clear upper quotient
DIV CX ;Divide by 10
OR DL,30H ;Store ASCII character
MOV [SI],DL
DEC SI
JMP B20
B30: OR AL,30H ;Store last quotient
MOV [SI],AL ; as ASCII character
RET
B10CONV ENDP
END BEGIN

```

شکل ۶-۱۴ تبدیل اعداد باینری به قالب اسی

159

```

page 60,132
TITLE A12MACRO (EXE) Simple macros
;-----
INITZ MACRO ;Define macro
MOV AX,@data ;Initialize segment
MOV DS,AX ; registers
MOV ES,AX ; registers
ENDM ;End macro

FINISH MACRO ;Define macro
MOV AX,4C00H ;End processing
INT 21H ;End macro
ENDM ;End macro
;-----
.MODEL SMALL
.STACK 64
.DATA
0000 54 65 73 74 20 6D MESSAGE DB 'Test macro instruction',13,10,'$'
61 63 72 6F 20 69
6E 73 74 72 75 63
74 69 6F 6E 0D 0A
24

;-----
.CODE
0000 BEGIN PROC FAR
INITZ ;Macro instruction
MOV AX,@data ;Initialize segment
MOV DS,AX ; registers
MOV ES,AX ; registers
MOV AH,09H ;Request display
LEA DX,MESSAGE ;Message
INT 21H
FINISH ;End macro
MOV AX,4C00H ;End processing
INT 21H ;End macro
ENDD
BEGIN END BEGIN

```

شکل ۹۲-۱ دستورات ماکرو اسمبل شده ساده

160



```

page 60,132
TITLE A12MACRO (EXE) Use of parameters
;-----
INITZ MACRO ;Define macro
MOV AX,@data ;Initialize segment
MOV DS,AX ; registers
MOV ES,AX ; registers
ENDM ;End macro

PROMPT MACRO MESSAGE ;Define macro
MOV AH,09H ;Request display
LEA DX,MESSAGE ; prompt
INT 21H ; prompt
ENDM ;End macro

FINISH MACRO ;Define macro
MOV AX,4C00H ;End processing
INT 21H ;End macro
ENDM ;End macro
;-----
.MODEL SMALL
.STACK 64
.DATA
0000 43 75 73 74 6F 6D MESSAGE1 DB 'Customer name?', '$'
65 72 20 6E 61 6D
65 3F 24
000F 43 75 73 74 6F 6D MESSAGE2 DB 'Customer address?', '$'
65 72 20 61 6A 64
72 65 73 73 3F 24

;-----
.CODE
0000 BEGIN PROC FAR
INITZ ;Initialize segment
MOV AX,@data ; registers
MOV DS,AX ; registers
MOV ES,AX ; registers
PROMPT MESSAGE ;Request display
MOV AH,09H ;Request display
LEA DX,MESSAGE2 ; prompt
INT 21H ; prompt
FINISH ;End macro
MOV AX,4C00H ;End processing
INT 21H ;End macro
ENDD
BEGIN ENDD
END BEGIN

```

شکل ۹۲-۲ استفاده از پارامترهای ماکرو

161



```

page 60,132
TITLE A12MACRO (EXE) Use of LOCAL
;-----
INITZ MACRO ;Define macro
MOV AX,@data ;Initialize segment
MOV DS,AX ; registers
MOV ES,AX ; registers
ENDM ;End macro

DIVIDE MACRO DIVIDEND, DIVISOR, QUOTIENT
LOCAL COUNT
LOCAL LOOP
LOCAL OUT
; AX = div's BX = divisor, CX = quotient
MOV AX,DIVIDEND ;set dividend
MOV BX,DIVISOR ;set divisor
SUB CX,CX ;clear quotient

COMP:
CMP AX,BX ;dividend = divisor?
JB OUT ; yes, exit
SUB AX,BX ;dividend - divisor
INC CX ;add to quotient
JMP COMP

OUT:
MOV QUOTIENT,CX ;store quotient
ENDM ;End macro

FINISH MACRO ;Define macro
MOV AX,4C00H ;End processing
INT 21H ;End macro
ENDM ;End macro
;-----
.MODEL SMALL
.STACK 64
.DATA
DIVDND DW 150 ;dividend
DIVSOR DW 27 ;divisor
QUOTNT DW ? ;quotient
;-----
BEGIN PROC FAR
INITZ ;Initialize segment
MOV AX,@data ; registers
MOV DS,AX ; registers
MOV ES,AX ; registers
DIVIDE DIVDND,DIVSOR,QUOTNT
; AX = div's BX = divisor, CX = quotient
MOV AX,DIVIDEND ;set dividend
MOV BX,DIVSOR ;set divisor
SUB CX,CX ;clear quotient

; 770000
CMP AX,BX ;dividend = divisor?
JB OUT ; yes, exit
SUB AX,BX ;dividend - divisor
INC CX ;add to quotient
JMP 770000

; 770001
MOV QUOTNT,CX ;store quotient
CALL 770000
FINISH ;End macro
ENDD
BEGIN ENDD
END BEGIN

```

شکل ۹۲-۳ استفاده از پارامترهای ماکرو LOCAL

162

