

باسمہ تعالیٰ

جزوہ زبان ماشین و اسمبلی

مدرس: رضا سعیدی نیا

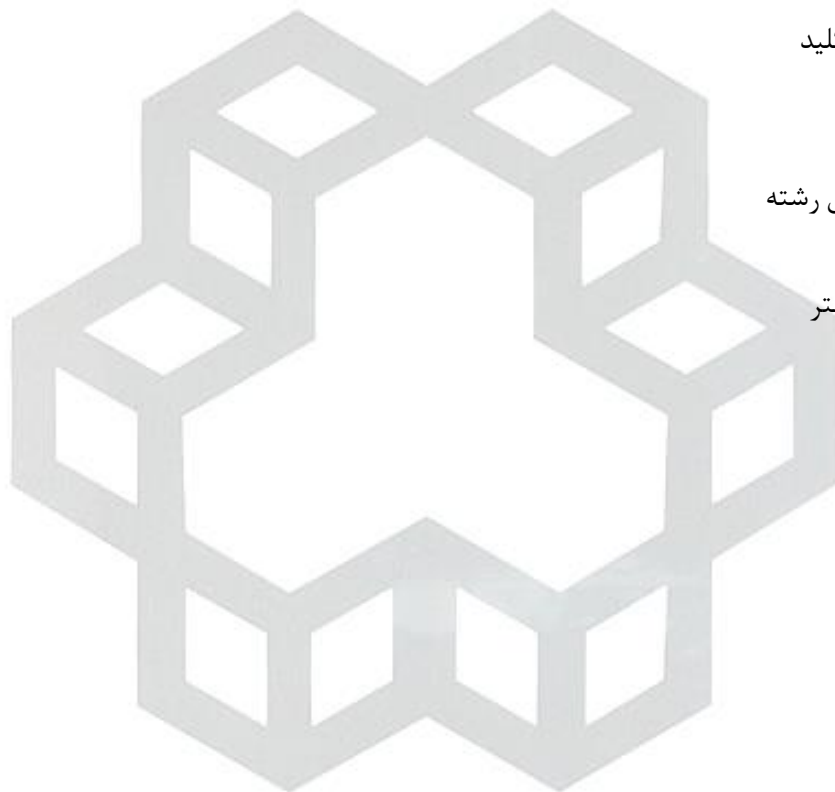
دانشگاہ صنعتی خواجه نصیر الدین طوسی

فهرست

3	مقدمه
4	فصل اول
4	تبدیل مبنا و عملیات روی اعداد
5	روش های ذخیره سازی اعداد صحیح در کامپیوتر
7	روش ذخیره سازی اعداد اعشاری در کامپیوتر (ممیز شناور)
7	محاسبات در مبنای R
8	فصل دوم
8	مقدمه ای بر سخت افزار
9	مجموعه ثبات پردازنده 8086
11	فصل سوم
11	تعریف سگمنت
11	قالب کلی برنامه اسمبلی
12	تعریف متغیر
14	دستورات اسمبلی
16	انواع عملوند ها
18	فصل چهارم
18	وقفه
19	تشریح صفحه نمایش
21	مدل های حافظه
23	فصل پنجم
23	دستورات محاسباتی
27	فصل ششم
27	شرط و حلقه
27	مقایسه
27	پرش
29	حلقه
32	دستورات منطقی
33	دستورات شیفت

زبان ماشین و اسمبلی

34	دستورات دوران
38	فصل هفتم
38	توابع و زیر برنامه
44	فصل هشتم
44	فایل های اجرایی
45	فصل نهم
45	مدیریت صفحه کلید
50	فصل دهم
50	پردازش رشته
50	دستورات پردازش رشته
55	ضمیمه
63	منابع مطالعه بیشتر
63	ارتباط



دانشگاه صنعتی خواجه نصیرالدین طوسی



[رابطه سخت افزار و زبان های برنامه سازی]

دستور زیر به که به زبان C نوشته شده است را در نظر بگیرید:

```
x = x * y + z - 2;
```

این دستور به زبان اسمبلی شامل چند دستور خواهد شد که باید اعمال زیر را انجام دهد:

```
CPU <- x
CPU <- x * y
CPU <- x * y + z
CPU <- x * y + z - 2
x <--- x * y + z - 2
```

مقایسه زبان های سطح بالا و اسمبلی:

1. زبان های سطح بالا به نوشتار نزدیک است.
2. زبان های اسمبلی تعداد خطوط بیشتری دارد.
3. سرعت اسمبلی بعلت حذف سر بار کامپایلر و کنترل مستقیم سخت افزار بیشتر است.
4. زبان های سطح بالا برای طراحی محیط های کاربر پسند و زبان اسمبلی برای کاربردهای نیازمند سرعت و کنترل سخت افزار کاربر دارد.



تبدیل مبنا و عملیات روی اعداد

تبدیل از مبنای 10 به ۲: قسمت صحیح را با تقسیم های متوالی بر ۲ بدست می آوریم تا زمانی که خارج قسمت بر ۲ قابل تقسیم نباشد. سپس از آخرین خارج قسمت به چپ باقی مانده ها را می نویسیم. قسمت اعشار را با ضرب متوالی در ۲ و یادداشت قسمت صحیح حاصل ضرب بدست می آوریم تا زمانی که قسمت اعشار صفر شود یا به دقت مورد نظر برسیم.

مثال:

$$(12.75)_{10} = (?)_2 = (1100.11)_2$$

$$12 = (1100)_2$$

$$0.75$$

$$\times 2$$

$$1.5$$

$$\times 2$$

$$1.0$$

مثال:

$$(14.25)_{10} = (?)_2 = (1110.01)_2$$

$$0.25$$

$$\times 2$$

$$0.50$$

$$\times 2$$

$$1.0$$

تبدیل از مبنای ۲ به 10: قسمت صحیح را در توان های مثبت ۲ و قسمت اعشار را در توان های منفی ۲ ضرب می کنیم و حاصل را جمع کرده، بدست می آوریم.

$$(a_n a_{n-1} \dots a_1 a_0 . a_{-1} \dots a_{-m})_r = a_n \times r^n + a_{n-1} \times r^{n-1} + \dots + a_1 \times r^1 + a_0 \times r^0 + a_{-1} \times r^{-1} + \dots + a_{-m} \times r^{-m} = \sum_{i=-m}^n a_i r^i$$

مثال:

$$(1110.01)_2 = (?)_{10} = 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} = (14.25)_{10}$$

تبدیل از مبنای 2 به 16: از ممیز به سمت چپ در قسمت صحیح چهار بیت چهار بیت جدا می کنیم و به جای هر چهار بیت رقم معادل آن را در مبنای 16 می نویسیم. از ممیز به سمت راست در قسمت اعشار چهار بیت چهار بیت جدا می کنیم و به جای هر چهار بیت معادل آن را می نویسیم.

مثال:

$$(0110 \ 1011.1101 \ 0100)_2 = (6B.D4)_{16}$$

$$(0100,1101,0001.0010)_2 = (4D1.2)_h$$

$$(0010,1101,0011,1001.0110,1000)_2 = (2D39.68)_{16}$$

در اسمبلی برای نمایش مبنای 16 بعد از عدد حرف h، مبنای 2 از حرف b و مبنای 8 از حرف 0 استفاده می کنیم. البته سمت چپ ترین رقم باید عدد باشد نه حرف.

زبان ماشین و اسمبلی

مبنای 10	مبنای 2	مبنای 16	مبنای 8
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	8	10
9	1001	9	11
10	1010	A	12
11	1011	B	13
12	1100	C	14
13	1101	D	15
14	1110	E	16
15	1111	F	17

[جدول تناظر سه مبنای 2، 10 و 16]

تبدیل از مبنای 16 به 2: به جای هر رقم مبنای 16 از جدول چهار بیت معادل آن را قرار می دهیم.

مثال:

$$(2E05A.0F)_{16} = (0010\ 1110\ 0000\ 0101\ 1010.0000\ 1111)_2$$

تبدیل از مبنای 2 به 8: از ممیز به سمت چپ در قسمت صحیح سه بیت سه بیت جدا می کنیم و به جای هر سه بیت رقم معادل آنرا در مبنای 8 می نویسیم. از ممیز به سمت راست در قسمت اعشار سه بیت سه بیت جدا می کنیم و به جای هر سه بیت معادل آن را می نویسیم.

تبدیل از مبنای 8 به 2: به جای هر رقم مبنای 8 از جدول سه بیت معادل آن را قرار می دهیم.

تبدیل از مبنای 16 به 8 و بالعکس: ابتدا به مبنای 2 برده سپس به مبنای مقصد می بریم.

مثال:

$$\begin{aligned} (5EA23.1C)_{16} &= (?)_2 = (?)_8 \\ (5EA23.1C)_{16} &= (0101\ 1110\ 1010\ 0010\ 0011.0001\ 1100)_2 \\ &= (001\ 011\ 110\ 101\ 000\ 100\ 011.000\ 111)_2 = (1365043.07)_8 \end{aligned}$$

مثال:

$$\begin{aligned} (1E250.21)_{16} &= (?)_2 = (?)_8 \\ (1E250.21)_{16} &= (0001\ 1110\ 0010\ 0101\ 0000.0010\ 0001)_2 \\ &= (011\ 110\ 001\ 001\ 010\ 000.001\ 000\ 010)_2 = (361120.102)_8 \end{aligned}$$

روش های ذخیره سازی اعداد صحیح در کامپیوتر

برای ذخیره اعداد بدون علامت از نمایش مبنای 2 استفاده می شود. در این حالت در n بیت از 0 تا $2^n - 1$ جا می شود. برای ذخیره اعداد علامت دار سه روش داریم:

زبان ماشین و اسمبلی

1. روش علامت - مقدار (sign-magnitude): سمت چپ ترین بیت عدد را علامت در نظر می گیریم برای اعداد مثبت صفر و برای اعداد منفی یک قرار می دهیم. باقی بیت های عدد مقدار آن است.
مثال:

$$\begin{aligned} +12 &= \underline{00001100} \\ -12 &= \underline{10001100} \\ +0 &= \underline{00000000} \\ -0 &= \underline{10000000} \end{aligned}$$

در این روش در n بیت از $(2^{n-1} - 1)$ تا $2^{n-1} - 1$ جا می شود.
این روش دو اشکال دارد؛ اول آنکه برای 0 دو نمایش منفی (در هشت بیت: 10000000) و مثبت (در هشت بیت: 00000000) وجود دارد و دوم آنکه برای انجام عمل تفریق مدار جداگانه لازم است، زیرا بدون مدار جداگانه بعضی اعمال تفریق به درستی انجام نمی شود.

مثال:

$$\begin{aligned} +12 &= 00001100 \\ -12 &= \underline{10001100} + \\ &= 10011000 = -24 \end{aligned}$$

2. روش متمم یک (1's complement): در این روش عدد مثبت مثل قبل است ولی عدد منفی متمم یک عدد مثبت است. برای محاسبه متمم یک، یک ها به صفر و صفر ها به یک تبدیل می شوند.
مثال:

$$\begin{aligned} +12 &= \underline{00001100} \\ -12 &= \underline{11110011} \end{aligned}$$

در این روش در n بیت از $(2^{n-1} - 1)$ تا $2^{n-1} - 1$ جا می شود.
این روش نیز یک اشکال دارد؛ آنکه برای 0 هنوز هم دو نمایش منفی (در هشت بیت: 11111111) و مثبت (در هشت بیت: 00000000) وجود دارد. اشکال دوم روش اول با این روش رفع می شود.
3. روش متمم دو (2's complement): در این روش عدد مثبت مثل قبل می باشد و عدد منفی متمم دو عدد مثبت است. برای بدست آوردن متمم دو یک عدد از روش زیر استفاده می کنیم:

$$1 + \text{متمم یک} = \text{متمم دو}$$

و یا عدد را از راست به چپ پیمایش می کنیم تا به اولین یک برسیم تا اولین یک را نگه می داریم و از آنجا به بعد را **not** می کنیم.
مثال:

$$\begin{aligned} +12 &= \underline{00001100} \\ -12 &= \underline{11110100} \end{aligned}$$

در این روش در n بیت از $2^{n-1} - 1$ تا $2^{n-1} - 1$ جا می شود.
این روش هر دو اشکال روش های قبل را بر طرف می کند.

زبان ماشین و اسمبلی

روش ذخیره سازی اعداد اعشاری در کامپیوتر (ممیز شناور)

ابتدا عدد اعشاری را نرمال می کنیم، یعنی ممیز را به سمت چپ ترین رقم مخالف صفر منتقل می کنیم. عدد به دست آمده شامل دو قسمت کسر و نما خواهد بود که بخش هایی هستند که در حافظه ذخیره می شوند. چون هر پردازنده از مبنای خاصی استفاده می کند نیازی به ذخیره آن نیست.

بیست و چهار بیت کسر	هفت بیت نما	بیت علامت کسر
---------------------	-------------	---------------

[نمایشی از ذخیره عدد اعشاری در حافظه]

مثال:

$$\begin{aligned}
 +(154.212)_{10} &= +(0.154212 \times 10^{+3}) \\
 -(2135.02)_{10} &= -(0.213502 \times 10^{+4}) \\
 +(0.00543)_{10} &= +(0.543 \times 10^{-2}) \\
 -(0.00012)_{10} &= -(0.12 \times 10^{-3})
 \end{aligned}$$

در این روش بیت علامت مخصوص کسر است و نما علامت ندارد. برای تشخیص نمای مثبت و منفی از هم بطور پیش فرض یک عدد پایه در نما قرار می گیرد (مثلا 64 یا 32) و نما با آن جمع می شود.

مثال:

$$+(110100110.1101)_2 = +(0.1101001101101) \times 2^{+9}$$

0	1001001	110100110110100000000000
---	---------	--------------------------

$$-(11011.111101)_2 = -(0.11011111101) \times 2^{+5}$$

1	1000101	110111111010000000000000
---	---------	--------------------------

$$+(0.00111)_2 = +(0.111) \times 2^{-2}$$

0	0111110	111000000000000000000000
---	---------	--------------------------

$$+(FE20.02)_{16} = +(0.FE2002) \times 16^{+5}$$

45	FE2002
----	--------

محاسبات در مبنای ۲

محاسبات عددی در هر مبنایی مانند مبنای 10 انجام می شود. صرفا باید توجه داشت که در جمع رقم نقلی (رقمی که در جمع از مرتبه پایین به مرتبه بالا منتقل می شود) و رقم قرضی (رقمی که در تفریق از مرتبه بالا به مرتبه پایین منتقل می شود) ارزشی برابر با مبنای عدد یعنی ۲ دارند.

مثال: دو عدد 1A25Bh و 19FC6h در مبنای 16 داده شده اند. عمل جمع و تفریق را روی آنها انجام دهید.

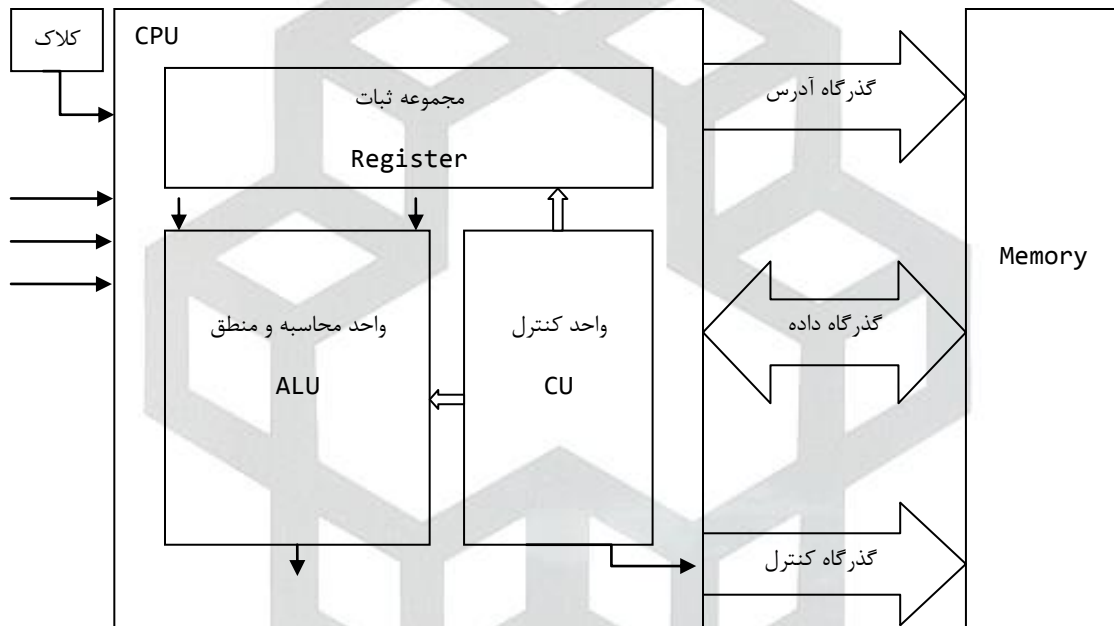
$ \begin{array}{r} 1A25B \\ 19FC6 \\ - \\ \hline 00295 \end{array} $	$ \begin{array}{r} 1A25B \\ 19FC6 \\ + \\ \hline 34331 \end{array} $
--	--



مقدمه ای بر سخت افزار

اکثر پردازنده های فعلی یا Intel هستند و یا سازگار با آن. AMD نیز با Intel سازگار است، معماری کامپیوتر متفاوت دارند ولی دستورات مشابه اجرا می کنند.

سیر پردازنده های Intel: 8086, 80186, 80286, 80386, 80486, 80586 یا Pentium, I, II, III, IV, multi-Core, V و ...



[ساختار پردازنده]

کلاک (ساعت): تعیین سرعت پردازنده. این پایه بطور متناوب صفر و یک می شود. هر بار تناوب صفر و یک شدن را یک پالس ساعت می گویند. هر پردازنده می تواند در هر پالس ساعت یک دستور اجرا کند (بعضی ها کمتر بعضی بیشتر). واحد سرعت ساعت Hz است. $1 \text{ GHz} = 10^9 \text{ Hz}$ یعنی ساعت در ثانیه یک میلیارد بار پالس می زند.

واحد محاسبه و منطق: وظیفه محاسبات ریاضی و منطقی را دارد.

واحد کنترل: وظایف سایر واحد ها را مشخص می کند.

واحد مجموعه ثبات: ثبات یک حافظه موقت داخل پردازنده است. سریع ترین حافظه است.

گذرگاه آدرس: تعیین کننده آدرس خانه حافظه است. هر چه تعداد خطوط آدرس بیشتر باشد، حجم حافظه قابل پشتیبانی بیشتر است. مثلاً 8086، بیست خط آدرس داشت. یعنی $2^{20} = 1 \text{ MB}$ حافظه پشتیبانی می کرد. $8 \text{ GB} = 2^{33}$, $16 \text{ GB} = 2^{34}$.

زبان ماشین و اسمبلی

گذرگاه داده: تعداد بیت های داده قابل انتقال در هر آن واحد (پالس ساعت) بین پردازنده و حافظه گذرگاه داده است. 8086. شانزده خط داده داشت.

مجموعه ثبات پردازنده 8086

این پردازنده چهار نوع ثبات 16 بیتی دارد:

1. **ثبات های عمومی:** در اکثر دستورات اسمبلی قابل استفاده اند:

- **ax (accumulator register):** معمولاً دستورات اسمبلی نتایج خود را در ثبات ax قرار می دهند. این

ثبات از دو بخش 8 بیتی به نام al و ah تشکیل شده است.

- **bx (base register):** در آدرس دهی ها می تواند بخش آدرس یک متغیر را نگه دارد و از دو بخش 8 بیتی

به نام b1 و bh تشکیل شده است.

- **cx (counter register):** در حلقه ها نقش شمارنده را دارد و از دو بخش 8 بیتی به نام cl و ch تشکیل

شده است.

- **dx (data register):** در محاسبات بزرگ قسمت با ارزش محاسبه در dx قرار می گیرد و از دو بخش 8

بیتی به نام dl و dh تشکیل شده است.

2. **ثبات های سگمنت:** هر برنامه اسمبلی حداکثر چهار بخش دارد و آدرس شروع این بخش ها در ثبات های سگمنت قرار دارد:

- **cs (code segment):** آدرس شروع بخش کد را نگه می دارد.

- **ds (data segment):** آدرس شروع بخش داده را نگه می دارد.

- **ss (stack segment):** آدرس شروع بخش پشته را نگه می دارد.

- **es (extra segment):** آدرس بخش اضافی را نگه می دارد.

3. **ثبات های کنترل و وضعیت:** وضعیت سایر بخش ها را کنترل می کنند:

- **ip (instruction pointer):** شماره خطی که پردازنده باید در مرحله بعد اجرا کند را نگه می دارد. مقدار

ip میتواند افزایش یا کاهش داشته باشد.

آدرس: مکان دقیق فیزیکی خط برنامه را مشخص می کند.

آفست: تفاوت مکان نسبت به محل شروع می باشد.

آدرس یک خط برنامه اسمبلی به صورت **CS:ip** مشخص می شود و با روش زیر محاسبه می شود:

آدرس 20 بیتی = $cs0 + ip$

نکته: مقدار ثبات CS در 10h ضرب می شود زیرا CS مقداری 16 بیتی را نگه می دارد و این در حالی است که گذرگاه آدرس، 20 بیتی است.

مثال: فرض کنید در CS عدد 12E0h و در ip عدد 200Ah را داریم آدرس خط برنامه چیست؟

$$cs0 = 12E00$$

$$ip = \underline{200A} + 14E0A$$

- **flags:** به معنای پرچم است، وضعیت محاسبات ALU را نگه می دارد.

12-15	o	d	i	t	s	z		a		p		C
-------	---	---	---	---	---	---	--	---	--	---	--	---

[نمایشی از ثبات پرچم]

. **cf (carry flag):** آخرین بیت خروجی از سمت چپ را نگه می دارد که به آن رقم نقلی می گویند.

زبان ماشین و اسمبلی

- pf (parity flag):** توازن را مشخص می کند و به منظور تشخیص خطا استفاده می شود:
- توازن زوج (**even parity**): تعداد یک‌ها در داده و توازن زوج است.
 - توازن فرد (**odd parity**): تعداد یک‌ها در داده و توازن فرد است.
- af (auxiliary carry flag):** رقم نقلی خروجی را چهار بیت پایین به چهار بیت بالا را نگه می دارد.
نکته: به هر نیم بایت یک nibble می گویند.
- zf (zero flag):** اگر نتیجه محاسبات صفر شود، این بیت یک می شود و در غیر این صورت صفر است.
- sf (sign flag):** اگر نتیجه محاسبات منفی شود، این بیت یک می شود و در غیر این صورت صفر است.
- tf (trap flag):** اگر این بیت یک باشد اجرای برنامه خط به خط انجام می شود.
- if (interrupt flag):** اگر یک باشد پردازنده می خواهد به وقفه ها پاسخ دهد و در غیر این صورت صفر است.
- df (direction flag):** جهت پردازش رشته را مشخص می کند. اگر صفر باشد پردازش از چپ به راست و اگر یک باشد از راست به چپ است.
- of (overflow flag):** اگر نتیجه محاسبات از محدوده خارج شود و سرریز رخ دهد این بیت یک می شود.
4. **ثبات های اندیس:** برای اندیس گذاری در آرایه ها و رشته ها استفاده می شود.
- **si (source index):** در آدرس دهی ها می تواند نقش اندیس داشته باشد. در دستورات پردازش رشته آفست رشته منبع را نگه می دارد.
 - **di (destination index):** در آدرس دهی ها می تواند نقش اندیس داشته باشد. در دستورات پردازش رشته آفست رشته مقصد را نگه می دارد.
 - **sp (stack pointer):** آفست بالای پشته را نگه می دارد.
 - **bp (base pointer):** در آدرس دهی ها می تواند نقش اندیس داشته باشد. همراه با **sp** برای ارسال (دریافت) پارامتر به (از) زیر برنامه استفاده می شود.
- در پردازنده های 32 بیتی به بعد، ثبات های 32 بیتی **esp, edi, esi, ebp, ebx, ecx, edx, eax** به سیستم اضافه شدند.

فصل سوم: دستورات ساده اسمبلی



فایل اسمبلی با فرمت `asm`، در سیستم ذخیره می شود.

تعریف سگمنت

نام سگمنت `segment` [پارامتر تنظیم] [پارامتر ترکیب] [پارامتر کلاس]

بدنه سگمنت

نام سگمنت `ends`

نام سگمنت، شناسه است. شناسه باید با رعایت اصول زیر همراه باشد:

1. اولین کاراکتر آن باید حرف باشد.
2. از کاراکترهای خاص مانند `/`، `?`، `\`، `*` و ... در آن استفاده نشود.
3. از کلمات رزرو نباشد.

پارامتر تنظیم: طریقه قرار گرفتن سگمنت در حافظه را مشخص می کند و یکی از مقادیر زیر را می تواند بگیرد:

1. `byte`: سگمنت هر جای حافظه می تواند قرار بگیرد.
2. `word`: آدرس شروع سگمنت باید زوج باشد.
3. `para`: آدرس شروع سگمنت باید بر 16 قابل تقسیم باشد. حالت پیش فرض.
4. `page`: آدرس شروع سگمنت بر 256 قابل تقسیم باشد.

پارامتر ترکیب: طریقه ترکیب بخش های مشابه در حافظه را مشخص می کند و می تواند یکی از مقادیر زیر را داشته باشد:

1. `none`: سگمنت هم نام وجود ندارد.
2. `public`: سگمنت های مشابه را در حافظه پشت سر هم قرار می گیرند. حالت پیش فرض.
3. `common`: برای سگمنت های مشابه حافظه مشترک در نظر می گیرد.
4. `stack`: سگمنت نوع پشته است.

پارامتر کلاس: نوع سگمنت را مشخص می کند و می تواند یکی از مقادیر زیر را داشته باشد:

1. `'code'`: از نوع سگمنت کد است.
2. `'data'`: از نوع سگمنت داده است.
3. `'stack'`: از نوع سگمنت پشته است.
4. `'extra'`: از نوع سگمنت اضافی است.

قالب کلی برنامه اسمبلی

```
eseg segment 'extra'
```

تعریف متغیر

```
eseg ends
```

```
sseg segment 'stack'
```

```
sseg ends
```

```
dseg segment 'data'
```

تعریف متغیر

```
dseg ends
```

زبان ماشین و اسمبلی

```
cseg segment 'code'
main proc far
    بدنه تابع
main endp
تعریف توابع دیگر
cseg ends
end main
```

نکته: در اسمبلی در هر خط یک دستور و یا یک تعریف متغیر نوشته می شود. برای اضافه کردن توضیحات از ; استفاده می شود.
نکته: برای تعریف سگمنت روش دیگری هم وجود دارد که در فصل بعد، تحت عنوان مدل های حافظه معرفی می شود.

تعریف متغیر

نام متغیر نوع مقدار یا ?

نام متغیر نیز شناسه است و باید از اصول مذکور درباره شناسه تبعیت کند.
نوع متغیر اندازه حافظه ای که به آن تخصیص داده می شود را تعیین می کند.
مقدار متغیر می تواند مجهول و یا معلوم باشد؛ می تواند رشته، کاراکتر و یا عدد باشد.
دستور تعریف انواع متغیر عبارتند از:

1. `(define byte) db`: تعریف متغیر یک بایتی:

مثال:

x1	db	10	x1	10
x2	db	?	x2	?
x3	db	'a'	x3	'a'
x4	db	15, 10, 20, 30	x5	'h'
x5	db	"hello"	x5+1	'e'
			x5+2	'l'
			x5+3	'l'
			x5+4	'o'

نکته: تعریف رشته هایی بدون محدودیت طول فقط با دستور `db` امکان پذیر است. برای تعریف رشته در دستور های دیگر باید طول رشته برابر میزان حافظه ای باشد که به متغیر اختصاص داده می شود.

2. `(define word) dw`: تعریف متغیر دو بایتی:

مثال:

w1	dw	5000
w2	dw	-500, 400, 120
w3	dw	1010111100001100b

3. `(define double word) dd`: تعریف متغیر چهار بایتی:

مثال:

d1	dd	200000	d2	20h
d2	dd	1E20A520h	d2	A5h
d3	dd	0ABE200h	d2	20h
d4	dd	-500, 250000, 345000	d2	1Eh

زبان ماشین و اسمبلی

pi dd 3.14

d4	-500
d4+4	250000
d4+8	345000

4. dq (define quad word): تعریف متغیر هشت بایتی:

مثال:

q1 dq 0
q2 dq ?

5. dt (define ten word): تعریف متغیر ده بایتی:

مثال:

t1 dt ?

6. equ (equal): تعریف مقدار ثابت:

مثال:

p equ 3.14

7. textequ (text equal): تعریف رشته ثابت:

مثال:

str1 textequ "hello students"

8. dup (duplicate): تعریف آرایه:

نوع نام آرایه (مقدار اولیه یا ؟) تعداد dup

مثال:

```
a1 db 100 dup(0)
a2 dw 100 dup(?)
a3 db 50 dup(0,1)
;repeat 0 and 1, 50 times
a4 dw 100 dup(0,5 dup(1))
;size a4 = 100 × 2 × 6 = 1200B
a5 dd 50 dup(3 dup(1),4 dup(2))
;size a5 = 50 × 7 × 4 = 1400B
```

a1	0
a1+1	0
:	:
a1+99	0
:	:
a3	0
a3+1	1
a3+2	0
a3+3	1
:	:
a3+98	0
a3+99	1

نکته: اگر سمت چپ ترین رقم مبنای 16 حرف باشد، باید قبل از آن 0 گذاشت تا اسمبلر فرق آن با متغیری با همین نام را درک کند.

نکته: برای ذخیره سازی اعداد در حافظه دو روش big endian و little endian وجود دارد. نام گذاری این دو روش به نام های مذکور به داستان گالیور و لی لی پوت ها مربوط می شود که گروهی در آنها معتقد بودند سمت بزرگ تخم مرغ

زبان ماشین و اسمبلی

سر آن است (big endian) و گروه دیگر معتقد بودند که سمت کوچک تخم مرغ سر آن است (little endian) و به همین علت با هم جنگ داشتند. در حال حاضر از روش دوم استفاده می شود. در این روش، ذخیره سازی عدد در حافظه از بایت کم ارزش به بایت پر ارزش صورت می گیرد. در واقع بخش کوچکتر بالاتر و بخش بزرگتر پایین تر قرار می گیرد.

دستورات اسمبلی

1. **mov (move)**: مقدار عملوند دوم را در عملوند اول می ریزد.

mov عملوند دوم و عملوند اول

مثال:

```
mov ax,10 ;ax = 10
mov ax,bx
mov x,20
```

نکته: دستور **mov** استثنائاتی دارد:

- دو عملوند نمی توانند محل حافظه باشند.
- طول دو عملوند باید برابر باشد.
- دو عملوند نمی توانند ثبات سگمنت باشند.
- با این دستور نمی توان ثبات **flags** را مقدار داد.

2. **lea (load effective address)**: آفست متغیر را در ثبات ذخیره می کند:

lea متغیر (خانه حافظه) و ثبات

مثال:

```
lea bx,a2 ;bx = offset of a2
```

نکته: دستور **offset** نیز آفست یک متغیر را مشخص می کند. ترکیب دستور **offset** و **mov** وظیفه **lea** را انجام می دهد.

نکته: توجه داشته باشید که آدرس یک متغیر مقداری 16 بیتی است و حتما باید در ثبات 16 بیتی منتقل شود.

مثال:

```
mov bx,offset a2
```

3. **assume**: این دستور برای مقدار دهی به ثبات سگمنت استفاده می شود و در اولین خط بعد از **main** باید نوشته شود (در روش فعلی تعریف سگمنت):

assume نام سگمنت اضافی: **es**، نام سگمنت پشته: **ss**، نام سگمنت داده: **ds**، نام سگمنت کد: **cs**

4. **inc (increment)**: به عملوند یک واحد اضافه می کند. برابر با عملگر **++** در **C++** است:

inc عملوند

مثال:

```
mov ax,10 ;store 10 in ax
inc ax ;ax = 11
```

5. **dec (decrement)**: از عملوند یک واحد کم می کند. برابر با عملگر **--** در **C++** است:

dec عملوند

مثال:

```
dec ax
```

زبان ماشین و اسمبلی

`dec x`

6. `xchg` (exchange): مقادیر دو عملوند را با هم عوض می کند:

`xchg` عملوند دوم و عملوند اول

مثال:

```
mov ax,20
mov bx,30
xchg ax,bx ;ax = 30 and bx = 20
```

7. `push`: مقدار عملوند را در بالای پشته قرار می دهد:

`push` عملوند

مثال:

`push ax`

8. `pop`: مقدار بالای پشته را در عملوند قرار می دهد:

`pop` عملوند

مثال:

`pop x`

9. `pusha` (`push all`): معادل هشت `push`. مقادیر ثابت های عمومی و اندیس را در پشته قرار می دهد.

10. `popa` (`pop all`): معادل هشت `pop`. مقادیر بالای پشته را در ثابت های عموم و اندیس قرار می دهد.

11. `pushf` (`push flags`): مقدار ثابت `flags` را در بالای پشته قرار می دهد.

12. `popf` (`pop flags`): مقدار بالای پشته را در ثابت `flags` قرار می دهد.

13. `pushad`: ثابت های `eax`, `ebx`, `ecx`, `edx`, `esi`, `ebp`, `esp` را در پشته قرار می دهد.

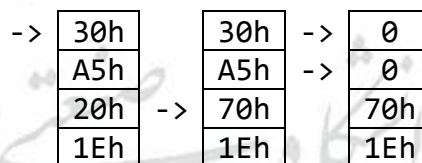
14. `popad`: مقادیر بالای پشته را در ثابت های فوق قرار می دهد.

15. `ptr` (`pointer`): برای دسترسی به قسمتی از متغیر و در سه حالت `byte`, `word` و `double word` به کار

میرود.

مثال:

```
mov byte ptr d2,30h ;access 1 byte
mov al,byte ptr d2+1 ;al = 0A5h
mov byte ptr d2+2,70h
mov word ptr d2,0 ;access 2 byte
mov ax,word ptr d2+2 ;ax = 1E70h
mov eax,double ptr x ;access 4 byte
```



16. `movzx` (`move zero extended`): برای انتقال عملوند کوچک به بزرگ؛ این دستور قسمت کم ارزش عملوند

بزرگ را با عملوند کوچک و قسمت پر ارزش آن را به صفر پر می کند.

مثال:

```
mov al,10010011b
movzx bx,al ;bx=0000000010010011b
```

17. `movsx` (`move sign extended`): برای انتقال عملوند کوچک به بزرگ؛ این دستور قسمت کم ارزش عملوند

بزرگ را با عملوند کوچک و قسمت پر ارزش آن را با علامت عملوند کوچک پر می کند.

زبان ماشین و اسمبلی

مثال:

```
mov  al,10010011b
movsx bx,al ;bx=1111111110010011b
```

انواع عملوند ها

1. عملوند بلا فصل

مثال:

```
mov  ax,10
mov  al,'a'
mov  bx,02A4h
```

2. عملوند ثبات

مثال:

```
mov  ax,bx
```

3. عملوند حافظه

مثال:

```
mov  eax,d2
```

4. عملوند غیر مستقیم ثبات

مثال:

```
lea  bx,a1
mov  [bx],10 ;a1 = 10
inc  bx
mov  [bx],25 ;a1+1 = 25
```

5. عملوند مستقیم + آفست

مثال:

```
mov  [a1+1],25
mov  [a1+2],25
...
mov  [a1+99],25
;a1 is direct operand and 1, 2, ..., 99 are offset
```

6. عملوند مستقیم + پایه یا اندیس

مثال:

```
mov  bx,1
mov  [a1+bx],25 ;a1 is direct operand, bx is base
mov  a1[bx],25 ;same as top statement
```

7. عملوند پایه و اندیس

مثال:

```
lea  bx,a1
mov  si,1
mov  [bx+si],25 ;bx is base and si is index
```

8. عملوند پایه + اندیس + آفست

زبان ماشین و اسمبلی

مثال:

```
lea bx,a1
mov si,0
mov [bx+si+1],25
mov al,[bx+si+1]
```

نکته: بجای پایه از هر یک از ثبات های **bx** و **bp** می توان استفاده کرد، همچنین به جای اندیس از هر یک از ثبات های **si** و **di** می توان استفاده کرد؛ ولی هیچ گاه نمی توان دو ثبات پایه و یا دو ثبات اندیس را با یکدیگر به کار برد.

مثال: متغیر **a1** که در زیر تعریف شده است چگونه در حافظه ذخیره می شود؟ تکه کدی بنویسید که بایت اول آن را در **al** قرار دهد و دو بایت دوم را در **bx**.

```
a1 dd 12E52052h
```

متغیر بر اساس روش **little endian** در حافظه ذخیره می شود:

```
mov al,byte ptr a1 ;a1 = 52h
mov bx,word ptr a1+2 ;bx = 12E5h
```

a1	52h
a1+1	20h
a1+2	E5h
a1+3	12h

مثال: مقدار حافظه مورد تخصیص داده شده به متغیر **a2** که به صورت زیر تعریف شده است چقدر است؟

```
a2 dq 100 dup (5 dup(0),2 dup(1))
```

متغیر، از نوع هشت بایتی است. به تعداد 100 بار، 5 عدد 0 و 2 عدد 1 در حافظه قرار می گیرد؛ بنابراین:

$$100 \times (5 + 2) \times 8 = 5600 \text{ B}$$

دانشگاه صنعتی خواجه نصیرالدین طوسی

فصل چهارم: وقفه و دستگاه های ورودی/خروجی ساده



وقفه

پردازنده خانواده 80x86، تعداد 256 وقفه را پاسخ می دهد که با شماره های 0 تا 255 مقدار دهی شده اند. آدرس شروع برنامه های پاسخ به وقفه در جدولی به نام جدول بردار وقفه (interrupt vector table) قرار دارد. معمولاً این جدول در خانه صفر حافظه قرار می گیرد. آدرس شروع هر برنامه 4 بایت است؛ بنابراین 1 KB اندازه جدول بردار وقفه است. وقتی یک شماره وقفه درخواست داده می شود مکان آدرس شروع برنامه آن بصورت زیر محاسبه می شود:

با دستور زیر وقفه فراخوانی می شود:

`int` شماره وقفه

هر برنامه از تعدادی تابع تشکیل شده است. می توانیم شماره تابع را مشخص کنیم. شماره تابع در ثبات `ah` یا `ax` قرار می گیرد.

مثال:

```
;call function 2 of interrupt 10h
mov ah,2
int 10h
```

قبل از فراخوانی وقفه باید ورودی های مورد نیاز را از طریق ثبات (ها) به وقفه ارسال کنیم. خروجی ها توسط تابع در ثبات (ها) برگردانده می شوند.

خواندن کاراکتر: تابع شماره `1h` وقفه `21h` یک کاراکتر را از صفحه کلید خوانده و در ثبات `al` قرار می دهد. کاراکتر مورد نظر در صفحه نمایش نشان داده می شود.

مثال:

```
mov ah,1h
int 21h
```

نکته: تابع `8h` وقفه `21h` نیز همین عمل را انجام می دهد ولی کاراکتر در صفحه نمایش نشان داده نمی شود.

مثال:

```
mov ah,8h
int 21h
```

نمایش کاراکتر: تابع `2h` وقفه `21h` یک کاراکتر موجود در ثبات `dl` را نمایش می دهد.

مثال:

```
mov dl,'a'
mov ah,2h
int 21h
```

چاپ رشته: تابع `9h` وقفه `21h` رشته ای را چاپ می کند که آفست متغیر آن در `dx` است. تا زمانی که به کاراکتر `$` برسد.

مثال:

```
msg1 db "hello students", "$"
lea dx,msg1
mov ah,9h
int 21h
```

زبان ماشین و اسمبلی

خواندن رشته: تابع `0Ah` وقفه `21h` رشته ای را می خواند و در متغیری می ریزد که آفست آن در `dx` است. متغیر مورد نظر باید بصورت زیر تعریف شود:

نام رشته `label byte`

مقدار طول `db` حداکثر طول
 ? `db` متغیر طول
 ('\$' یا '?') `dup` مقدار طول `db` نام آرایه

مثال:

```
str1 label byte
max db 40
len db ? ;store length
str1_1 db 40 dup (?) ;store string
...
lea dx, str1
mov ah, 0Ah
int 21h
```

تشریح صفحه نمایش

مانیتور در دو حالت کار می کند:

- مد گرافیک: نقاط مانیتور `pixel` نامیده می شود.
 - مد متن: مانیتور شامل 25 سطر و 80 ستون است. در تقاطع هر سطر و ستون یک حرف نوشته می شود. بنابراین تعداد دویست حرف را نشان می دهد. هر حرف دو بایت اشغال می کند که یک بایت آن کد اسکی حرف و یک بایت آن صفت (رنگ) می باشد. پس هر صفحه 4 KB حافظه اشغال می کند. آدرس شروع مد متن در `B8000(0)h` می باشد. مانیتور چهار صفحه دارد که در کل 16 KB فضا اشغال می کنند. شماره صفحات از 0 تا 3 مقدار می گیرند. همیشه بطور پیش فرض صفحه 0 نمایش داده می شود. آدرس نقاط از `(0,0)` تا `(24,79)` می باشد.
- صفت هر حرف یک بایت است که چهار بایت کم ارزش آن مشخص کننده رنگ کاراکتر و چهار بایت پر ارزش آن مشخص کننده رنگ زمینه و حالت چشمک زن است:

سه بیت رنگ	بیت میزان روشنایی	سه بیت رنگ	بیت چشمک زن
------------	-------------------	------------	-------------

[نمایش الگوی صفت کاراکتر]

سیاه	0000	بنفش	0101
آبی	0001	قهوه ای	0110
سبز	0010	سفید	0111
کبود	0011	خاکستری	1000
قرمز	0100	آبی پررنگ	1001

[جدول کد رنگ ها]

جابجایی مکان نما: تابع `2h` وقفه `10h` باعث جابجایی مکان نما به مکان `(x,y)` می شود که مختصات آن به ترتیب در `dh` و `dl` و شماره صفحه در `bh` قرار گرفته باشد.

زبان ماشین و اسمبلی

مثال:

```
mov dh,20
mov dl,20
mov bh,0
mov ah,2h
int 10h
```

پاک کردن مانیتور: تابع 6h وقفه 10h از مکان (x_1, y_1) تا مکان (x_2, y_2) را پاک می کند که مختصات نقطه شروع به ترتیب در dh و dl، مختصات نقطه پایان به ترتیب در ch و cl، تعداد خطوط در al و صفت در bh قرار گرفته باشد.

مثال:

```
mov dh,10
mov dl,10
mov ch,20
mov cl,20
mov al,11
mov bh,7
mov ah,6h
int 10h
```

نمایش کاراکتر به صورت رنگی: تابع 9h وقفه 10h حرفی را که در al قرار گرفته، به صفت و تعدادی که به ترتیب در bl و cx مشخص شده و در صفحه ای که شماره اش در bh قرار گرفته است، در مانیتور نشان می دهد. این تابع مکان نما را جابجا نمی کند.

مثال:

```
mov al,'A'
mov ah,9h
mov cx,20
mov bl,11110100b
mov bh,0
int 10h
```

نکته: تابع 0Ah وقفه 10h نیز یک حرف را به تعداد دلخواه به رنگی که در حال حاضر فعال شده است چاپ می کند. تمامی ورودی ها مثل تابع 9h است ولی مشخصه رنگ ندارد.

مثال: برنامه ای بنویسید که ابتدا صفحه نمایش پاک کند سپس، با چاپ پیام مناسب یک حرف را از کاربر دریافت کند و آنرا در وسط مانیتور به تعداد بیست بار و به رنگ قرمز روی زمینه آبی چاپ کند.

```
dseg segment 'data'
msg1 db "please enter a character", "$"
harf db ?
dseg ends
cseg segment 'code'
main proc far
assume cs:cseg,ds:dseg
mov ax,dseg
mov ds,ax
;*****clear screen*****
```

زبان ماشین و اسمبلی

```
mov     dh,0
mov     dl,0
mov     ch,24
mov     cl,79
mov     al,25
mov     bh,7
mov     ah,6h
int     10h
;*****print msg1*****
lea     dx,msg1
mov     ah,9h
int     21h
;*****read character*****
mov     ah,1h
int     21h
mov     harf,al
;*****goto (12,40)*****
mov     dh,12
mov     dl,40
mov     bh,0
mov     ah,2h
int     10h
;*****print harf*****
mov     al,harf
mov     cx,20
mov     bl,00010100b
mov     bh,0
mov     ah,9h
int     10h
;*****
mov     ah,1h
int     21h
mov     ax,4C00h
int     21h
main    endp
cseg   ends
end     main
```

مدل های حافظه

مدل های حافظه روشی است برای تعیین میزان حافظه مصرفی هر سگمنت:

نوع مدل .model

انواع مدل حافظه عبارتند از:

.tiny .1: داده و کد کمتر 64 KB

زبان ماشین و اسمبلی

2. small: کد کمتر یا برابر 64 KB، داده کمتر یا برابر 64 KB.
 3. medium: کد بزرگتر 64 KB، داده کمتر یا برابر 64 KB.
 4. compact: کد کمتر یا برابر 64 KB، داده بزرگتر 64 KB.
 5. large: کد بزرگتر 64 KB، داده بزرگتر 64 KB.
 6. huge: کد بزرگتر 64 KB، داده بزرگتر 64 KB (یک داده به تنهایی میتواند 64 KB باشد).
 7. flat: سگمنت ندارد، وارد مد حفاظت شده سیستم می‌شود.
- با استفاده از این روش می‌توان سگمنت‌ها را به شیوه زیر تعریف کرد:

نوع سگمنت .

مثال:

```
.stack 1000
.data
.code
main proc far
    mov     ax,@data
    mov     ds,ax
main endp
end main
```

نکته: برای استفاده از دستورات یک پردازنده خاص (بالتر از 8086) باید از دستور زیر پیش از شروع سگمنت کد استفاده کرد:

شماره پردازنده .

مثال: برای استفاده از دستورات پردازنده 80386 باید کد زیر پیش از شروع سگمنت کد قرار گیرد:

.386

دانشگاه صنعتی خواجه نصیرالدین طوسی

فصل پنجم: دستورات محاسباتی



دستورات محاسباتی

تمامی دستورات محاسباتی بر ثبات **flags** تاثیر می گذارند.

دستورات محاسباتی عبارتند از:

1. **add (addition)**: دو عملوند را جمع می کند و حاصل را در عملوند اول می ریزد:

add عملوند دوم و عملوند اول

مثال:

```
mov ax,200
mov bx,300
add ax,bx ;ax = 500
```

2. **adc (addition with carry)**: دو عملوند و مقدار **cf** را جمع می کند و حاصل را در عملوند اول می ریزد:

adc عملوند دوم و عملوند اول

3. **sub (subtract)**: عملوند دوم را از عملوند اول کم می کند و حاصل را در عملوند اول می ریزد:

sub عملوند دوم و عملوند اول

مثال:

```
mov ax,200
mov bx,300
sub ax,bx ;ax = -100
```

4. **sbb (subtract with borrow)**: مقدار **cf** را از حاصل تفریق دو عملوند کم می کند و حاصل را در عملوند او می ریزد:

sbb عملوند دوم و عملوند اول

5. **mul (multiply)**

mul عملوند

- اگر عملوند یک بایتی باشد آن را در **al** ضرب کرده و حاصل را در **ax** قرار می دهد.

مثال:

```
mov al,20
mov bl,50
mul bl ;ax = 1000
```

- اگر عملوند دو بایتی باشد آن را در **ax** ضرب کرده و حاصل را در **dx:ax** قرار می دهد.

مثال:

```
mov ax,200
mov bx,500
mul bx ;dx:ax = 100000
```

- اگر عملوند چهار بایتی باشد آن را در **eax** ضرب کرده و حاصل را در **edx:eax** قرار می دهد.

مثال:

```
x dd 240000
y dd 351545
```


زبان ماشین و اسمبلی

```
z    dq    ?
mov  eax,y
mul  x
mov  double ptr z,eax
mov  double ptr z+4,edx
```

نکته: `dx:ax` مقادیر را به صورت چهار بایتی ذخیره می کند، بنابراین باید دقت داشت ارقامی که در `dx` قرار خواهند گرفت، رقم 16 تا 32 را تشکیل می دهند و در محاسبات ارزش آنها باید در نظر گرفته شود. این مسئله درباره `edx:eax` بعنوان محل ذخیره هشت بایتی نیز صادق است.

نکته: دستور `cbw (convert byte to word)` ثابت `ax` را به `ax` تبدیل می کند (ضرب یک بایت در دو بایت).

مثال:

```
mov  al,25
mov  bx,300
cbw
mul  bx ;dx:ax = 7500
```

نکته: دستور `cwd (convert word to double word)` ثابت `ax` را به `dx:ax` تبدیل می کند (ضرب دو بایت در چهار بایت).

مثال:

```
x    dw    2450
y    dd    100000
z    dq    0
mov  ax,x
cwd
mul  y
```

نکته: دستور `imul (integer multiply)` برای انجام عمل ضرب صحیح در یک مقدار ثابت به کار می رود و علاوه بر سه روش گفته شده برای دستور `mul` می تواند به شکل های زیر نیز به کار رود:

`imul` عملوند (ثابت)

`imul` عملوند دوم * عملوند اول = عملوند اول ; عملوند دوم (ثابت), عملوند اول (ثابت)

`imul` عملوند سوم * عملوند دوم = عملوند اول ; عملوند سوم (ثابت), عملوند دوم (ثابت 16 بیتی), عملوند اول (ثابت 32 بیتی)

نکته: دستورات گفته شده برای `imul` در پردازنده 80386 اضافه شده اند و برای استفاده از آنها باید این نکته را در نظر داشت.

مثال:

```
imul ax,10
imul ebx,cx,200
```

6. (divide) div:

`div` عملوند

- اگر عملوند یک بایتی باشد، مقسوم در `ax` قرار می گیرد. خارج قسمت را در `al` و باقی مانده در `ah` قرار می دهد.

مثال:

```
mov  ax,2520
mov  bl,25
div  bl ;al = 100, ah = 20
```

زبان ماشین و اسمبلی

مثال:

```
x    db    130
y    db    13
...
mov  al,x
cbw
div  y ;al = 10, ah = 0
```

- اگر عملوند دو بایتی باشد، مقسوم در `dx:ax` قرار می گیرد. خارج قسمت را در `ax` و باقی مانده در `dx` قرار می دهد.

مثال:

```
mov  dx,0
mov  ax,2000
mov  bx,120
div  bx ;ax = 16, dx = 80
```

- اگر عملوند چهار بایتی باشد، مقسوم در `edx:eax` قرار می گیرد. خارج قسمت را در `eax` و باقی مانده در `edx` قرار می دهد.

مثال:

```
y    dq    12FAC55014A0142Ah
x    dd    241E0014h
...
mov  eax,double ptr y ;eax = 14A0142Ah
mov  edx,double ptr y+4 ;edx = 12FAC550h
div  x
```

نکته: دستور `idiv` (integer divide) برای انجام عمل تقسیم صحیح بر یک مقدار ثابت به کار می رود و علاوه بر سه روش گفته شده برای دستور `div` می تواند به شکل های زیر به کار رود:

`idiv` عملوند (ثابت)

`idiv` عملوند دوم / عملوند اول = عملوند اول; عملوند دوم (ثابت)، عملوند اول (ثبات)

نکته: دستورات گفته شده برای `idiv` در پردازنده 80386 اضافه شده اند و برای استفاده از آنها باید این نکته را در نظر داشت.

مثال:

```
idiv ax,20
```

مثال: فرض کنید `bx = 150`، `ax = 1500` و `dx = 2` باشد. آنگاه با اجرای دستور زیر مقادیر `ax` و `dx` چه تغییری می کند؟

```
div  bx
```

باید توجه داشت که `dx:ax` حاوی عدد 21500 نیست، زیرا 2 موجود در `dx` رقم 16 ام عدد بوده و باید ارزش آن نیز مد نظر باشد، عدد موجود در `dx:ax` برابر است با:

$$2 \times 2^{16} + 1500 = 132572$$

بنابراین 132572 بر مقدار `bx` یعنی 150 تقسیم می شود که در نتیجه آن خارج قسمت در `ax` و باقی مانده در `dx` ذخیره می شود؛ پس `ax = 883` و `dx = 122`.

فصل ششم: پرش، شرطها، حلقه ها و دستورات منطقی



شرط و حلقه

در C یا C++ داریم:

if (شرط)

{دستورات}

در واقع دستور شرطی **if** به این صورت عمل می کند که اگر شرط مورد نظر برقرار نباشد به بیرون **if** پرش می شود و گرنه دستورات اجرا می شود.

while (شرط)

{دستورات}

در یک حلقه **while** اگر شرط برقرار نباشد به بیرون **while** پرش می شود و گرنه دستورات اجرا می شود و در انتها به ابتدای **while** پرش می شود.

if (شرط)

{دستورات اول}

else

{دستورات دوم}

در یک دستور شرطی **if else** اگر شرط برقرار نباشد به ابتدای **else** پرش می شود و دستورات مربوطه اجرا می شود و گرنه دستورات **if** اجرا می شود، سپس به بیرون **if else** پرش می شود.

حال باید دستورات شرطی، کنترلی و حلقه را به همین ترتیب در زبان اسمبلی پیاده سازی کنیم.

مقایسه:

در دستور مقایسه عملوند اول از عملوند دوم کم می شود، ولی حاصل ذخیره نخواهد شد و صرفاً ثبات **flags** تحت تاثیر قرار خواهد گرفت. در مقایسه نتایج بر روی **zf**، **sf** و یا **cf** تاثیر می گذارد و نتیجه بسته به علامت دار بودن و نبودن عملوند ها در نظر گرفته می شود:

cmp عملوند دوم، عملوند اول

داده بدون علامت

Above

Below

equal to

داده علامت دار

greater than >

less than <

equal to =

[جدول مقایسه]

پرش

برای پرش نیازمند برچسب هستیم:

نام برچسب:

دو نوع پرش وجود دارد:

1. پرش بدون شرط (**unconditional jump**): بدون بررسی کردن هیچ شرطی به برچسب پرش می کند:

jmp نام برچسب

زبان ماشین و اسمبلی

2. پرش شرطی (conditional jump): یک شرط را چک می کند، اگر شرط درست بود پرش اتفاق می افتد و گرنه خط بعد اجرا می شود. پرش های شرطی بر دو نوع می باشند. پرش برای داده های بدون علامت و پرش برای داده های علامت دار.

- پرش داده های علامت دار: هر جفت دستوری که با هم آمده اند، هم ارز اند:

1. اگر عملوند اول از عملوند دوم بزرگتر است، به برچسب پرش می کند و گرنه خط بعد:

`jg` نام برچسب `;jump if greater than`
`jnle` نام برچسب `;jump if not less than or equal to`

2. اگر عملوند اول از عملوند دوم بزرگتر مساوی است به برچسب پرش می کند و گرنه خط بعد:

`jge` نام برچسب `;jump if greater than or equal to`
`jnl` نام برچسب `;jump if not less than`

3. اگر عملوند اول از عملوند دوم کوچکتر است به برچسب پرش می کند و گرنه خط بعد:

`jl` نام برچسب `;jump if less than`
`jnge` نام برچسب `;jump if not greater than or equal to`

4. اگر عملوند اول از عملوند دوم کوچکتر مساوی است به برچسب پرش می کند و گرنه خط بعد:

`jle` نام برچسب `;jump if less than or equal to`
`jng` نام برچسب `;jump if not greater than`

5. اگر عملوند اول با عملوند دوم مساوی است به برچسب پرش می کند و گرنه خط بعد:

`je` نام برچسب `;jump if equal to`
`jz` نام برچسب `;jump if zero`

6. اگر عملوند اول با عملوند دوم نامساوی است به برچسب پرش می کند و گرنه خط بعد:

`jne` نام برچسب `;jump if not equal to`
`jnz` نام برچسب `;jump if not zero`

- پرش داده های بدون علامت: هر جفت دستوری که با هم آمده اند، هم ارز اند:

1. اگر عملوند اول از عملوند دوم بزرگتر است به برچسب پرش می کند و گرنه خط بعد:

`ja` نام برچسب `;jump if above`
`jnb` نام برچسب `;jump if not below or equal to`

2. اگر عملوند اول از عملوند دوم بزرگتر مساوی است به برچسب پرش می کند و گرنه خط بعد:

`jae` نام برچسب `;jump if above or equal to`
`jnb` نام برچسب `;jump if not below`

3. اگر عملوند اول از عملوند دوم کوچکتر است به برچسب پرش می کند و گرنه خط بعد:

`jb` نام برچسب `;jump if below`
`jnae` نام برچسب `;jump if not above or equal to`

4. اگر عملوند اول از عملوند دوم کوچکتر مساوی است به برچسب پرش می کند و گرنه خط بعد:

`jbe` نام برچسب `;jump if below or equal to`
`jna` نام برچسب `;jump if not above`

5. اگر عملوند اول با عملوند دوم مساوی است به برچسب پرش می کند و گرنه خط بعد:

زبان ماشین و اسمبلی

`je` نام برچسب `;jump if equal to`

`jz` نام برچسب `;jump if zero`

6. اگر عملوند اول با عملوند دوم نامساوی است به برچسب پرش می کند و گرنه خط بعد:

`jne` نام برچسب `;jump if not equal to`

`jnz` نام برچسب `;jump if not zero`

- دیگر دستورات پرش:

1. اگر $cf = 1$ پرش می کند و گرنه خط بعد:

`jc` نام برچسب

2. اگر $cf = 0$ پرش می کند و گرنه خط بعد:

`jnc` نام برچسب

3. اگر $sf = 1$ پرش می کند و گرنه خط بعد:

`js` نام برچسب

4. اگر $sf = 0$ پرش می کند و گرنه خط بعد:

`jns` نام برچسب

5. اگر $of = 1$ پرش می کند و گرنه خط بعد:

`jo` نام برچسب

6. اگر $of = 0$ پرش می کند و گرنه خط بعد:

`jno` نام برچسب

7. اگر $pf = 1$ پرش می کند و گرنه خط بعد:

`jp` نام برچسب

8. اگر $pf = 0$ پرش می کند و گرنه خط بعد:

`jnp` نام برچسب

9. اگر $cx = 0$ پرش می کند و گرنه خط بعد:

`jcxz` نام برچسب

حلقه: دستور `loop` برای پیاده سازی حلقه بکار می رود و از `CX` بعنوان شمارنده استفاده می کند. این دستور ابتدا از `CX` یکی کم

می کند، اگر حاصل صفر شد از حلقه خارج می شود و گرنه به برچسب مورد نظر پرش می کند.

مثال: حلقه با ده بار تکرار.

```
mov cx,10
```

```
for1:
```

```
بدنه حلقه
```

```
loop for1
```

مثال: حلقه زیر با وجود صفر بودن `CX` دستورات را اجرا می کند، چرا که ابتدا از آن کم کرده و بعد آن را بررسی می کند.

```
mov cx,0
```

```
for2:
```

```
بدنه حلقه
```

```
loop for2 ;perform loop for 65536 times
```

زبان ماشین و اسمبلی

مثال: کد اصلاح شده حلقه فوق. حلقه اجرا نمی شود.

```
mov cx,0
for3:
jcxz efor3
بدنه حلقه
loop for3
efor3:
```

مثال: حلقه بی نهایت.

```
mov cx,1
for4:
بدنه حلقه
inc cx
loop for4
```

مثال: برای کار کردن با cx در حلقه نیاز به push و pop کردن مقدار cx داریم که روی شمارش تاثیر گذاشته نشود.

```
mov cx,number
for5:
jcxz efor5
push cx
بدنه حلقه
pop cx
loop for5
efor5:
```

نکته: برای حلقه های بزرگتر از دستور loopd و از ecx بعنوان شمارنده استفاده می شود.

مثال:

```
mov ecx,1000000
for6:
بدنه حلقه
loopd for6
```

مثال: برنامه ای بنویسید که با چاپ پیام رشته ای را دریافت کرده و حروف کوچک آن را به بزرگ تبدیل و چاپ کند.

```
dseg segment
msg1 db "please enter a string:$"
msg2 db 10,13,"upper case is:$"
str1 label byte
max1 db 40
len1 db ?
str1_1 db 40 dup('$'),' '$
pkey db 10,13,"press any key...$"
dseg ends
sseg segment
dw 128 dup(0)
sseg ends
```

زبان ماشین و اسمبلی

```
cseg segment
main proc far
    assume     cs:cseg,ds:dseg,ss:sseg
    mov     ax,dseg
    mov     ds,ax
    ;***** print msg1 *****
    lea     dx,msg1
    mov     ah,9h
    int     21h
    ;***** read str1 *****
    lea     dx,str1
    mov     ah,0Ah
    int     21h
    ;***** print msg2 *****
    lea     dx,msg2
    mov     ah,9
    int     21h
    ;***** convert to uppercase **
    mov     cl,len1
    mov     ch,0
    mov     si,0
for1:  jcxz   efor1
    cmp     str1_1[si], 'a'
    jb     l1
    cmp     str1_1[si], 'z'
    ja     l1
    sub     str1_1[si],32
    l1:    inc     si
    loop   for1
efor1:
    ;***** print uppercase *****
    lea     dx,str1_1
    mov     ah,9h
    int     21h
    ;***** press any key *****
    lea     dx,pkey
    mov     ah,9h
    int     21h
    mov     ah,1h
    int     21h
    mov     ax,4C00h
    int     21h
main     endp
cseg     ends
```

زبان ماشین و اسمبلی

```
end main
```

دستورات منطقی

تمامی دستورات منطقی روی ثابت **flags** تاثیر دارند.

دستورات منطقی عبارتند از:

1. and: بیت های متناظر دو عملوند را **and** منطقی می کند و نتیجه را در عملوند اول ذخیره می کند:

and عملوند دوم, عملوند اول

مثال:

```
mov al,11001010b
mov bl,11110000b
and al,bl ;al = 11000000b
```

نکته: اگر بخواهیم بخشی از بیت های یک عملوند را صفر کنیم آن را با صفر **and** می کنیم. **and** با یک تاثیری روی بیت ندارد.

2. test: عمل دستور **and** را انجام می دهد ولی نتیجه ذخیره نمی شود بلکه صرفاً ثابت **flags** تحت تاثیر قرار می گیرد.

3. or: بیت های متناظر دو عملوند را **or** منطقی می کند و نتیجه را در عملوند اول ذخیره می کند:

or عملوند دوم, عملوند اول

مثال:

```
mov al,11001010b
mov bl,11110000b
or al,bl ;al = 11111010b
```

نکته: اگر بخواهیم بخشی از بیت های یک عملوند را یک کنیم آن را با یک **or** می کنیم. **or** با صفر تاثیری روی بیت ندارد.

4. xor: بیت های متناظر دو عملوند را با هم **xor** می کند و نتیجه را در عملوند اول قرار می دهد:

xor عملوند دوم, عملوند اول

مثال:

```
mov al,11001010b
mov bl,11110000b
xor al,bl ;al = 00111010b
```

نکته: اگر بخواهیم بخشی از بیت های یک عملوند را **not** کنیم آن را با یک **xor** می کنیم. **xor** با صفر تاثیری روی بیت ندارد.

5. not: متمم یک عملوند را می سازد:

not عملوند

مثال:

```
mov al,11001010b
not al ;al = 00111010b
```

6. neg: عملوند را در منفی ضرب می کند:

neg عملوند

دستورات شیفت

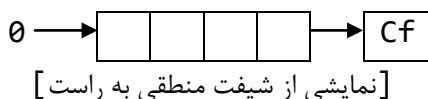
دو نوع شیفت داریم:

1. شیفت منطقی (logical shift):

shr - (shift to right): شیفت منطقی به راست. برای شیفت بیشتر از یک بار تعداد در **cl** قرار می گیرد:

زبان ماشین و اسمبلی

shr 1, عملوند
shr cl, عملوند



مثال:

```
mov al,11001010b
shr al,1 ; al = 01100101b and cf = 0
shr al,1 ; al = 00110010b and cf = 1
```

نکته: بعد از n بار شیفت به راست یک عملوند n بیتی صفر خواهد شد.

نکته: هر بار شیفت به راست معادل تقسیم صحیح بر 2 است و r بار شیفت به راست معادل تقسیم صحیح بر 2^r می باشد.

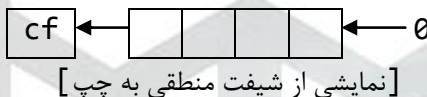
نکته: در داده های علامت دار شیفت منطقی به راست علامت را مثبت می کند.

مثال:

```
mov al,16 ; al = 00010000b
shr al,1 ; al = 00001000b = 8
shr al,3 ; al = al / 8
```

- shift to left shl: شیفت منطقی به چپ. برای شیفت بیشتر از یک بار تعداد در cl قرار می گیرد:

shl 1, عملوند
shl cl, عملوند



نکته: بعد از n بار شیفت به چپ یک عملوند n بیتی صفر خواهد شد.

نکته: هر بار شیفت به چپ معادل ضرب در 2 است و r بار شیفت به چپ معادل ضرب در 2^r می باشد.

مثال:

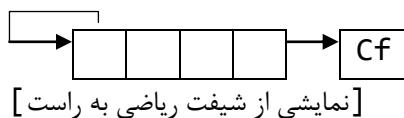
```
mov al,11001010b
shl al,1 ; al = 10010100b and cf = 1
mov al,16
shl al,1 ; al = 32
```

2. شیفت ریاضی (arithmetic shift):

- shift arithmetic to right sar: شیفت ریاضی به راست. برای شیفت بیشتر از یک بار تعداد در cl قرار

می گیرد:

sar 1, عملوند
sar cl, عملوند



مثال:

زبان ماشین و اسمبلی

```
mov al,11001010b
sar al,1 ; al = 11100101b and cf = 0
```

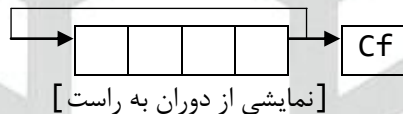
- (shift arithmetic to left) sal: شیفت ریاضی به چپ. همانند شیفت منطقی به چپ است. برای شیفت بیشتر از یک بار تعداد در cl قرار می گیرد:

```
sal 1, عملوند
sal cl, عملوند
```

دستورات دوران

1. (rotate to right) ror: دوران به راست. برای دوران بیشتر از یک بار تعداد در cl قرار می گیرد:

```
ror 1, عملوند
ror cl, عملوند
```



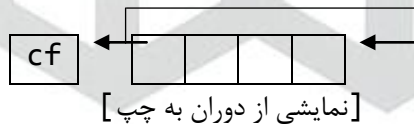
مثال:

```
mov al,11001010b
ror al,1 ; al = 01100101b and cf = 0
ror al,1 ; al = 10110010b and cf = 1
```

نکته: بعد از n بار دوران به راست یک عملوند n بیتی مقدار اولیه آن بدست می آید.

2. (rotate to left) rol: دوران به چپ. برای دوران بیشتر از یک بار تعداد در cl قرار می گیرد:

```
rol 1, عملوند
rol cl, عملوند
```

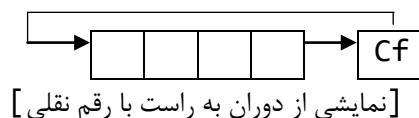


مثال:

```
mov al,11001010b
rol al,1 ; al = 10010101b and cf = 1
rol al,1 ; al = 00101011b and cf = 1
```

3. (rotate with carry to right) rcr: دوران به راست با رقم نقلی. برای دوران بیشتر از یک بار تعداد در cl قرار می گیرد:

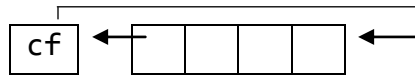
```
rcr 1, عملوند
rcr cl, عملوند
```



زبان ماشین و اسمبلی

4. rcl (rotate with carry to left): دوران به چپ با رقم نقلی. برای دوران بیشتر از یک بار تعداد در cl قرار می گیرد:

```
rcl    1,عملوند
rcl    cl,عملوند
```



[نمایشی از دوران به چپ با رقم نقلی]

مثال: عبارت $ax = 8bx - 16cx + 5$ را بدون استفاده از عملگر ضرب محاسبه کنید.

```
shl    bx,3 ;bx = 8bx
shl    cx,4 ;cx = 16cx
sub    bx,cx
add    bx,5
mov    ax,bx
```

مثال: عبارت $ax = 7bx + 17cx - 5$ را بدون استفاده از عملگر ضرب محاسبه کنید.

```
;7bx = 8bx - bx
;17cx = 16cx + cx
mov    si,bx
shl    bx,3
sub    bx,si ;bx = 7bx
mov    si,cx
shl    cx,4 ;cx = 16cx
add    cx,si ;cx = 17cx
add    bx,cx
sub    bx,5
mov    ax,bx
```

مثال: برنامه ای بنویسید که یک عدد را دریافت کرده و معادل آن را در مبنای 2 نمایش دهد.

```
dseg segment
msg1 db "please enter a digit:$"
msg2 db 10,13,"binary is:$"
str1 label byte
max db 6
len db ?
str1_1 db 6 dup('$')
number dw 0
ten dw 10
pkey db 10,13,"press any key...$"
dseg ends
sseg segment
dw 128 dup(0)
sseg ends
cseg segment
```

زبان ماشین و اسمبلی

```
main proc far
assume     cs:cseg,ds:dseg,ss:sseg
mov  ax,dseg
mov  ds,ax
lea  dx,msg1
mov  ah,9h
int  21h
lea  dx,str1
mov  ah,0Ah
int  21h
lea  dx,msg2
mov  ah,9h
int  21h
;*convert string to number
mov  cl,len
mov  ch,0 ;cx = len
mov  ax,0
mov  si,0
for1: jcxz efor1
      mov  bl,str1_1[si]
      sub  bl,48
      mov  bh,0
      mul  ten
      add  ax,bx
      inc  si
loop for1
efor1: mov  number,ax ;*convert to binary*****
      mov  cx,16
for2:  rol  number,1
      jc   l1
      mov  dl,'0'
      mov  ah,2h
      int  21h
      jmp  l2
l1:   mov  dl,'1'
      mov  ah,2h
      int  21h
      l2:
loop for2
      lea  dx,pkey
      mov  ah,9h
      int  21h
      mov  ah,1h
      int  21h
```

```
    mov ax,4C00h
    int 21h
main endp
cseg ends
end main
```



دانشگاه صنعتی خواجه نصیرالدین طوسی

فصل هفتم: توابع و زیربرنامه ها

توابع و زیر برنامه

تعریف تابع به صورت زیر انجام می پذیرد:

نام تابع `proc near/far`

بدنه تابع

`ret/retf`

نام تابع `endp`

برای فراخوانی یک تابع از دستور `call` استفاده می شود:

نام تابع `call`

تابع نوع near: تابع در سگمنت جاری قرار دارد. ثبات `cs` آدرس شروع سگمنت کد را نگه می دارد. ثبات `ip` شمارنده برنامه می باشد و تعیین کننده خط بعدی است که باید اجرا شود. دستور `call, ip` را در پشته قرار می دهد. و دستور `ret` دو بایت بالای پشته را بر می دارد و در `IP` قرار می دهد. بنابراین تعداد `push` و `pop` ها باید در داخل یک تابع برابر باشند.

تابع نوع far: تابع در سگمنت دیگری است. بنابراین دستور `call` چهار بایت `cs:ip` را در پشته قرار می دهد. و دستور `retf` مقدار چهار بایت بالای پشته را بر می دارد و در `cs:ip` قرار می دهد.

توابع ممکن است نیازمند ورودی و دارای خروجی باشند، دو روش برای ارسال/دریافت پارامتر به/از تابع وجود دارد:

1. استفاده از ثباتها: قبل از فراخوانی تابع پارامترها را توسط ثبات به تابع می فرستیم و تابع نیز نتیجه کار را در ثباتها برمی گرداند. متغیرهای تعریف شده در سگمنت داده سراسری هستند (`near`).

2. استفاده از پشته: قبل از فراخوانی تابع متغیرها را در پشته قرار می دهیم. در داخل تابع بدون برداشتن پارامترها از پشته بطور غیر مستقیم و توسط ثبات های `sp` و `bp` به پارامترها دسترسی پیدا می کنیم.

مثال: برنامه ای بنویسید که با استفاده از توابع ابتدا صفحه نمایش را پاک کند. سپس با چاپ یک پیام یک رشته دریافت کند و با استفاده از تابع حروف کوچک رشته را به بزرگ و بزرگ را به کوچک تبدیل کند و سپس نتیجه را چاپ کند.

```
dseg segment
msg1 db 10,13,"please enter a string:$"
msg2 db 10,13,"reverse is:$"
str1 label byte
max1 db 40
len1 db ?
str1_1 db 40 dup('$')
pkey db 10,13,"press any key...$"
dseg ends
cseg segment
main proc far
assume cs:cseg,ds:dseg
mov ax,dseg
mov ds,ax
mov es,ax
call clrscr
```

```

;*****
lea dx,msg1
call print
;*****
lea dx,str1
call read
;*****
lea dx,msg2
call print
;*****
mov cl,len1
mov ch,0 ;cx = len1
lea si,str1_1
call revrs
;*****
lea dx,str1_1
call print
;*****
lea dx,pkey
mov ah,9h
int 21h
mov ah,1h
int 21h
mov ax,4c00h
int 21h
cseg ends
print proc near
mov ah,9h
int 21h
ret
print endp
read proc near
mov ah,0Ah
int 21h
ret
read endp
clrscr proc near
mov ch,0
mov cl,0
mov dh,24
mov dl,79
mov al,25
mov bh,7
mov ah,6h

```

زبان ماشین و اسمبلی

```
int 10h
ret
clrscr endp
revrs proc near
frev:
jcxz efrev
cmp [si], 'A'
jb l1
cmp [si], 'Z'
ja l2
add [si], 32
jmp l1
l2:
cmp [si], 'a'
jb l1
cmp [si], 'z'
ja l1
sub [si], 32
l1:
inc si
loop frev
efrev:
ret
revrs endp
end main
```

چگونگی ارسال/دریافت پارامتر به/از پشته:

`push` پارامتر n ام

`push` پارامتر (n-1) ام

.

.

.

`push` پارامتر دوم

`push` پارامتر اول

`call` نام تابع

`pop` پارامتر اول

`pop` پارامتر دوم

.

.

.

`pop` پارامتر (n-1) ام

`pop` پارامتر n ام

زبان ماشین و اسمبلی

```

.
.
نام تابع proc near
push bp
mov bp,sp
mov [bp+4],10
add [bp+6],10
.
.
دستور عملوند, [bp+2n+2]
pop bp
ret
نام تابع endp

```

bp	<- sp
ip	
پارامتر اول	
پارامتر دوم	
.	
.	
ام (n-1) پارامتر	
پارامتر n	
.	
.	

و اگر تابع از نوع far باشد:

```

.
.
نام تابع proc far
push bp
mov bp,sp
mov [bp+6],10
add [bp+8],10
.
.
دستور عملوند, [bp+2n+4]
pop bp
ret
نام تابع endp

```

bp	<- sp
cs	
ip	
پارامتر اول	
پارامتر دوم	
.	
.	
ام (n-1) پارامتر	
پارامتر n	
.	
.	

مثال: برنامه ای بنویسید که یک عدد را دریافت کرده و اعداد زوج کوچکتر از آن را چاپ کند.

```

dseg segment
msg1 db 10,13,"please enter a digit:$"
str1 label byte
max1 db 6
len1 db ?
str1_1 db 6 dup('$')
str2 db 6 dup('$')

```

زبان ماشین و اسمبلی

```
ten    dw    10
n      dw    0
pkey   db    10,13,"press any key...$"
dseg   ends
sseg   segment
dw     128 dup(0)
sseg   ends
cseg   segment
main   proc far
assume cs:cseg,ds:dseg,ss:sseg
mov    ax,dseg
mov    ds,ax
mov    es,ax
lea    dx,msg1
call   print
lea    dx,str1
call   read
;*****
mov    cl,len1
mov    ch,0 ;cx = len1
lea    si,str1_1
call   atoi
mov    n,ax
;*****
w1:    cmp    ax,n
jge    ew1
push  ax
lea    si,str2
call   itoa
lea    dx,str2
call   print
pop    ax
add   ax,2
jmp   w1
ew1:   lea    dx,pkey
mov    ah,9h
int    21h
mov    ah,1h
int    21h
mov    ax,4C00h
int    21h
main   endp
print  proc near
```

```
    mov  ah,9h
    int  21h
    ret
print endp
read  proc near
    mov  ah,0Ah
    int  21h
    ret
read  endp
atoi proc near
    mov  ax,0
fatoi:
    jcxz efatoi
    mov  bl,[si]
    sub  bl,48
    mov  bh,0
    mul  ten
    add  ax,bx
    inc  si
    loop fatoi
efatoi:
    ret
atoi endp
itoa  proc near
    mov  cx,0
witoa:
    cmp  ax,0
    je   ewito
    mov  dx,0
    div  ten
    add  dx,48
    push dx
    inc  cx
    jmp  witoa
ewito:
fitoa:
    jcxz efitoa
    pop  dx
    mov  [si],dl
    inc  si
    loop fitoa
efitoa:
    mov  [si],'$'
    ret
```

```
itoa endp  
cseg ends  
end main
```

فصل هشتم: مقایسه فایل های exe, com



فایل های اجرایی

تفاوت بین فایل های exe و com:

1. هر دو فایل قابل اجرا توسط CPU هستند.
2. اولویت فایل com از exe بیشتر است.
3. فایل exe چهار سگمنت دارد ولی فایل com یک سگمنت. حجم فایل com کمتر از exe می باشد.
4. در ابتدای فایل com جدول 256 بایتی بنام جدول وضعیت برنامه (PSW) قرار دارد که مشخصات فایل مثل اندازه، نسخه سیستم عامل و ... در آن ذخیره می شود.
5. برنامه کاربر از خط 256 شروع می شود. برای تعیین شماره خط از دستور زیر استفاده می شود:

org شماره خط

org 256

مثال:

دانشگاه صنعتی خواجه نصیرالدین طوسی

فصل نهم: مدیریت صفحه کلید



مدیریت صفحه کلید

به ازای هر کلید دو بایت در بافر صفحه کلید ذخیره می شود که یک بایت آن کد اسکی و بایت دیگر آن کد پیمایش است. خواندن کلید از بافر: تابع 10h وقفه 16h دو بایت را از بافر صفحه کلید خوانده و در ثبات های ah و al قرار می دهد.

Ah	Al	نوع کلید
کد پیمایش	کد اسکی کلید	کلید معمولی
کد پیمایش	0	کلید توسعه یافته
کد پیمایش	0E0h	کلید توسعه یافته تکراری

[جدول تنظیمات دو بایت بافر صفحه کلید]

مثال:

```
mov ah,10h
int 16h
cmp al,0
jz keyE
cmp al,0E0h
je keyER
```

کلید معمولی

keyE:

```
cmp ah, کد پیمایش
```

```
je key
```

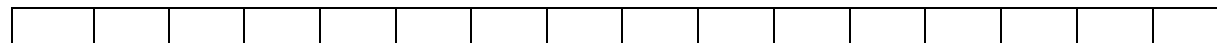
keyER:

```
cmp ah, کد پیمایش
```

```
je keyR
```

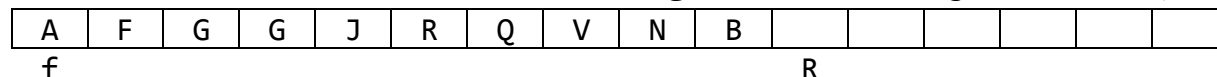
بافر صفحه کلید: بافر یک حافظه موقت برای ذخیره سازی داده ها است. این بافر یک آرایه 32 بایتی است و به صورت صف حلقوی سازمان دهی شده است. متغیر های front و rear به ترتیب سر صف و ته صف را نگه می دارند. در داخل بافر صفحه کلید پانزده عدد کلید ذخیره می شود.

آرایه ای با شانزده خانه در اختیار داریم و از آن بعنوان صف نام میبریم. هنگامی که پردازنده در حال پردازش و انجام عملیات باشد، کلید های وارد شده توسط کاربر در این صف ذخیره می شود. هنگامی که صف خالی است اشاره گر های rear و front که به ترتیب با r و f مشخص شده اند به خانه 0:



r=f

فرض کنید کاربر ده کلید از صفحه کلید را می فشارد، هر کلید در جلوی کلید دیگر و در محلی که r به آن اشاره می کند قرار می گیرد؛ همچنین f به محل خروج و یا همان سر صف اشاره می کند:



زبان ماشین و اسمبلی

در نظر بگیرید که پردازش های فعلی پردازنده به اتمام می رسد و چهار حرف از حروف قرار گرفته را از صف خارج می کند و مشغول به پردازش دیگری می شود. در این حالت چهار خانه ابتدایی صف خالی خواهد شد:

				J	R	Q	V	N	B					
				F				r						

این بار کاربر شش کلید را می فشارد، در این صورت اگر صف مانند یک آرایه معمولی باشد، دیگر کاربر قادر نخواهد بود که کلیدی را وارد کند، چرا که اشاره گر ته صف به انتهای صف یا به بیشترین مقدار مجاز رسیده است و این در حالیست که چهار خانه ابتدایی صف در حال حاضر خالی است. بنابراین پس از رسیدن به انتهای صف، اشاره گر R به خانه ابتدایی اشاره خواهد کرد:

				J	R	Q	V	N	B	A	C	J	S	U	O
				f				r							

اگر این صف شانزده حرف را بپذیرد با ورود چهار حرف دیگر پر خواهد شد، ولی مشکلی که وجود دارد این است که مقدار هر دو اشاره گر برابر می شود و این همان شرط خالی بودن صف است، در واقع پردازنده نمی تواند تشخیص دهد که صف پر است و یا خالی است:

D	R	H	M	J	R	Q	V	N	B	A	C	J	S	U	O
				f=r											

به همین دلیل در این صف فقط تا پانزده کاراکتر می پذیرد تا شرط پر و خالی بودن صف یکی نباشد:

D	R	H		J	R	Q	V	N	B	A	C	J	S	U	O
				R				f							

کلید ها	کد مبنای 16	کد مبنای 10
Shift + Tab	0fh	15
Alt + Q (W,E,R,T,Y,U,I,O,P)	10h-19h	16-25
Alt + A (S,D,F,G,H,J,K,L)	1eh-26h	30-38
Alt + Z (X,C,V,B,N,M)	2ch-32h	44-50
F1 - F10	3bh-44h	59-68
Mome	47h	71
up arrow	48h	72
Page Up	49h	73
left arrow	4bh	75
right arrow	4dh	77
End	4fh	79
down arrow	50h	80
Page Down	51h	81
Insert	52h	82
Delete	53h	83
Shift + F1 (F2,...,F10)	54h-5dh	84-93
Ctrl + F1 (F2,...,F10)	5eh-67h	94-103
Alt + F1 (F2,...,F10)	68h-71h	104-113
Ctrl + Prt Scr	72h	114
Ctrl + left arrow	73h	115

زبان ماشین و اسمبلی

Ctrl + right arrow	74h	116
Ctrl + End	75h	117
Ctrl + Page Down	76h	118
Ctrl + Home	77h	119
Alt + (2,3,4,5,6,7,8,9,0,')	78h-83h	120-131
Ctrl + Page Up	84h	132

[جدول کد های صفحه کلید توسعه یافته]

تست وجود کاراکتر: تابع 11h وقفه 16h تست می کند که آیا کلیدی در بافر وجود دارد یا خیر. اگر وجود داشته باشد zf = 0 و یک کپی از کلید در al و ah قرار می گیرد (کلید را از صف بر نمی دارد) و اگر بافر کلید نداشته باشد zf = 1. بایت وضعیت صفحه کلید: این بایت وضعیت کلیدهایی از صفحه کلید که باعث تغییر وضعیت کاراکترهای خوانده شده می شوند، مانند Shift, Ctrl Alt و ... را نگه می دارد. نکته: بایت های وضعیت در BIOS ذخیره شده اند، آدرس 40:17h بایت وضعیت صفحه کلید 83 کلیدی و 40:18h بایت وضعیت صفحه کلید 101 کلیدی است. خواندن بایت وضعیت صفحه کلید: تابع 12h وقفه 16h بایت وضعیت صفحه کلید 101 کلیدی را خوانده و در al قرار می دهد. همچنین تابع 2h وقفه 16h بایت وضعیت صفحه کلید 83 کلیدی را خوانده در al قرار می دهد.

Insert	Capslock	NumLock	Scrolllock	Alt	Ctrl	L.shift	R.Shift
7	6	5	4	3	2	1	0

[بایت وضعیت صفحه کلید 83 کلیدی]

Insert	Capslock	NumLock	Scrolllock	Ctrl+Numlock	SysRq	Alt	Ctrl
7	6	5	4	3	2	1	0

[بایت وضعیت صفحه کلید 101 کلیدی]

مثال:

```

mov ah,12h
int 16h
test al,10000000b ;10000000b = 80h
jz caps
;*print insert pressed*
...
cap:
test al,01000000b ;01000000b = 40h
jz num
;*print caps lock is on*
...
num:
test al,20h
jz scr
    
```

زبان ماشین و اسمبلی

```
;*print num lock is on*  
...  
scr:  
test al,10h  
jz     برچسب بعد  
;test can be done for 8h, 4h, 2h, 1h  
...
```

نکته: برای خواندن از دستگاه ورودی می توان از دستور **in** و برای نوشتن در دستگاه خروجی می توان از دستور **out** استفاده کرد. این دو دستور به صورت زیر به کار میروند:

in شماره پورت، عملوند
out عملوند، شماره پورت

```
in     al,2F8h  
out    2F8h,al  
;61h speaker system
```

مثال:

دانشگاه صنعتی خواجه نصیرالدین طوسی

فصل دهم: پردازش داده‌های رشته‌ای

پردازش رشته

به این خاطر که رشته‌ها در اسمبلی بسیار کاربردی بوده و بسیار از آنها استفاده می‌شود، دستوراتی مختص رشته‌ها به این زبان اضافه شده است. برای استفاده از این دستورات پیش نیازهای زیر وجود دارد:

1. متغیرها باید در سگمنت اضافی تعریف شوند؛ بنابراین یا باید سگمنت اضافی را تعریف کنیم یا آدرس مشترک به هر دو سگمنت داده و اضافی اختصاص دهیم، برای این منظور از مجموعه سه دستور زیر استفاده می‌کنیم:

```
mov ax,data
mov ds,ax
mov es,ax
```

2. آفست رشته مبدأ باید در **si** و آفست رشته مقصد باید در **di** ذخیره شود.

3. توسط **df** باید جهت پردازش رشته مشخص شود؛ اگر **df = 0** باشد پردازش از چپ به راست و اگر **df = 1** باشد پردازش از راست به چپ است. برای تغییر مقدار این بیت، دو دستور در اسمبلی وجود دارد:

```
std ;set df, df = 1
cld ;clear df, df = 0
```

4. شبه دستور **rep (repeat)** که به منظور تکرار دستورات رشته‌ای قبل از آنها قرار می‌گیرد و به تعداد مشخص شده در **CX** دستور را تکرار می‌کند:

```
mov cx,تعداد
rep دستور رشته ای
```

دستورات پردازش رشته

همه دستورات پردازش رشته می‌توانند در سه حالت **byte**، **word** و **double word** استفاده شوند:

1. **movs (move string)**: به سه صورت **movsb**، **movsw** یا **movsd** استفاده می‌شود و 1، 2 یا 4 بایت از رشته مبدأ که آفست آن در **si** است را به رشته مقصد که آفست آن در **di** است، کپی می‌کند. اگر **df = 0** باشد به **si** و **di** به مقدار 1، 2 یا 4 واحد اضافه می‌کند و اگر **df = 1** باشد به همین مقدار از آنها کم می‌کند.

مثال:

```
str1 db 100 dup('*')
str2 db 100 dup(?)
;for(i = 0,i < 100 ,i++) style
lea si,str1
lea di,str2
cld
mov cx,100 ;or cx,50 or cx,25
rep movsb ;*or movsw or movsd
;now str2 = str1 = '***...*'
;or code can be like this:
;for(i = 100 , i > 0 , i--) style
lea si,str1
add si,99
lea di,str2
```

str1	'*'
str1+1	'*'
str1+2	'*'
	.
	.
str1+99	'*'
	.
	.
str2	?
str2+1	?
str2+2	?
	.

زبان ماشین و اسمبلی

```
add di,99
std
mov cx,100
rep movsb
```

str2+99	.
	?

مثال: برنامه ای بنویسید که دو رشته را دریافت کند و رشته دوم را به انتهای رشته اول چسبانده و نتیجه را چاپ کند.

```
dseg segment
msg1 db 10,13,"please enter a string:$"
str1 label byte
max1 db 80
len1 db ?
str1_1 db 80 dup('$')
str2 label byte
max2 db 40
len2 db ?
str2_2 db 40 dup('$')
msg2 db 10,13,"merge of two strings is:$"
pkey db 10,13,"press any key...$"
dseg ends
cseg segment
main proc far
assume cs:cseg,ds:dseg,es:dseg
mov ax,dseg
mov ds,ax
mov es,ax
lea dx,msg1
mov ah,9h
int 21h
lea dx,str1
mov ah,0ah
int 21h
lea dx,msg1
mov ah,9h
int 21h
lea dx,str2
mov ah,0Ah
int 21h
lea si,str2_2
lea di,str1_1
mov cl,len1
mov ch,0
add di,cx
mov cl,len2
rep movsb
lea dx,msg2
```

زبان ماشین و اسمبلی

```
mov ah,9h
int 21h
lea dx,str1_1
mov ah,9h
int 21h
lea dx,pkey
mov ah,9h
int 21h
mov ah,1h
int 21h
mov ax,4C00h
int 21h
main endp
cseg ends
end main
```

2. `cmps` (`compare string`): به سه صورت `cmpsb`، `cmpsw` یا `cmpsd` استفاده می شود و 1، 2 یا 4 بایت از رشته مبدأ که آفست آن در `si` است را با رشته مقصد که آفست آن در `di` است، مقایسه می کند. در نتیجه این مقایسه ثبات `flags` تحت تاثیر قرار می گیرد. اگر `df = 0` باشد به `si` و `di`، به مقدار 1، 2 یا 4 واحد اضافه می کند و اگر `df = 1` باشد به همین مقدار از آنها کم می کند.

نکته: جهت بررسی برابری یا نابرابری دو رشته دو دستور `repe` و `repne` جایگزین دستور `rep` می شود. دستور `repe` مقایسه را تا زمانی تکرار می کند که کاراکترها برابر باشند و `CX` برابر صفر نباشد. به محض غلط شدن یکی از این دو شرط، حلقه تمام می شود. اگر `CX = 0` شود و حلقه تمام شود، یعنی دو رشته با هم کاملاً برابرند. دستور `repne` نیز مقایسه را تا زمانی تکرار می کند که کاراکترها برابر نباشند و `CX` برابر صفر نباشد. به محض غلط شدن یکی از این دو شرط، حلقه تمام می شود.

مثال:

```
lea si,str1
lea di,str2
mov cx,100
repe cmpsb
```

3. `scas` (`scan string`): به سه صورت `scasb`، `scasw` یا `scasd` استفاده می شود و `ax` یا `eax` را با رشته ای که آفست آن در `di` است، مقایسه می کند. اگر `df = 0` باشد به `di` به مقدار 1، 2 یا 4 واحد اضافه می کند و اگر `df = 1` باشد به همین مقدار از آن کم می کند.

مثال: برنامه ای بنویسید که دو رشته را دریافت کرده، آنها را مقایسه کند و در انتها پیام مناسبی چاپ کند.

```
.model small
.data
msg1 db 10,13,"please enter a string:$"
msg2 db 10,13,"equal$"
msg3 db 10,13,"not equal$"
str1 label byte
max1 db 40
```

```

len1 db      ?
str1_1 db    40 dup('$')
str2 label byte
max2 db      40
len2 db      ?
str2_2 db    40 dup('$')

.code
main proc far
mov ax,@data
mov ds,ax
mov es,ax
lea dx,msg1
mov ah,9h
int 21h
lea dx,str1
mov ah,0Ah
int 21h
lea dx,msg1
mov ah,9h
int 21h
lea dx,str2
mov ah,0Ah
int 21h
;*****
mov cl,len1
cmp cl,len2
jne l1
lea si,str1_1
lea di,str2_2
mov ch,0
repe cmpsb
jne l1
lea dx,msg2
mov ah,9h
int 21h
jmp l2
l1:
lea dx,msg3
mov ah,9h
int 21h
l2:
;*****
mov ax,4C00h
int 21h

```

زبان ماشین و اسمبلی

```
main endp  
end main
```

4. `lods` (`load_string`): به سه صورت `lods`، `lodsb` یا `lodsw` استفاده می شود و 1، 2 یا 4 بایت از رشته ای که آفست آن در `si` است را در `ax`، `ax` یا `eax` بار می کند. اگر `df = 0` باشد به `si` به مقدار 1، 2 یا 4 واحد اضافه می کند و اگر `df = 1` باشد به همین مقدار از آن کم می کند.

5. `stos` (`store_string`): به سه صورت `stosb`، `stosw` یا `stosd` استفاده می شود و `ax`، `ax` یا `eax` را در رشته ای که آفست آن در `di` است ذخیره می کند. اگر `df = 0` باشد به `si` به مقدار 1، 2 یا 4 واحد اضافه می کند و اگر `df = 1` باشد به همین مقدار از آن کم می کند.



دانشگاه صنعتی خواجه نصیرالدین طوسی

زبان ماشین و اسمبلی

چند نمونه تمرین:

1- برنامه ای بنویسید که یک عدد را دریافت کند مشخص کند اول هست یا خیر

```
data segment
    msg1      db          10,13,"please enter a digit:$"
    str1      label byte
    max1      db          6
    len1      db          ?
    str1_1    db          6 dup('$')
    str2      db          10,13,"digit is prime:$"
    str3      db          10,13,"digit is not prime$"
    ten       dw          10
    n         dw          0
    sum       dw          0
    pkey      db          10,13,"press any key...$"
ends
stack segment
    dw 128 dup(0)
ends
code segment
start:
    mov     ax,data
    mov     ds,ax
    mov     es,ax
    ;*****
    lea     dx,msg1
    call    print
    lea     dx,str1
    call    read
    ;*****
    mov     cl,len1
    mov     ch,0 ;cx = len1
    lea     si,str1_1
    call    atoi
    mov     n,ax
    ;*****
    mov     di,2
w1:
    cmp     di,n
    jge     ew1
    mov     ax,n
    mov     dx,0
    div    di
    cmp     dx,0
```

زبان ماشین و اسمبلی

```
    jne aval
    jmp naval
aval: inc di
jmp  w1
    ew1:
    lea dx, str2
    call print
    jmp l11
    naval:
    lea dx, str3
    call print
l11:
    lea    dx, pkey
    mov    ah, 9h
    int    21h
    mov    ah, 1h
    int    21h
    mov    ax, 4C00h
    int    21h
ends
print proc near
    mov    ah, 9h
    int    21h
    ret
print endp
read  proc near
    mov    ah, 0Ah
    int    21h
    ret
read  endp
atoi proc near
    mov    ax, 0
fatoi:
    jcxz  efatoi
    mov    bl, [si]
    sub    bl, 48
    mov    bh, 0
    mul    ten
    add    ax, bx
    inc    si
    loop  fatoi
efatoi:
    ret
atoi endp
itoa  proc near
```

زبان ماشین و اسمبلی

```
mov    cx,0
witoa:
cmp    ax,0
je     ewito
mov    dx,0
div    ten
add    dx,48
push  dx
inc    cx
jmp    witoa
ewito:
fitoa:
jcxz  efitoa
pop    dx
mov    [si],dl
inc    si
loop  fitoa
efitaa:
mov    [si],'$'
ret
itaa  endp
end start ; set entry point and stop the assembler.
```

2- مثال برنامه ای بنویسید که یک عدد را دریافت کند مشخص کند کامل است یا خیر؟

```
data segment
msg1   db 10,13,"please enter a digit:$"
str1   label byte
max1   db 6
len1   db ?
str1_1 db 6 dup('$')
str2   db 10,13,"digit is complete:$"
str3   db 10,13,"digit is not complete$"
ten    dw 10
n      dw 0
sum    dw 0
pkey   db 10,13,"press any key...$"
ends
stack segment
dw 128 dup(0)
ends
code segment
start:
mov    ax,data
mov    ds,ax
```


زبان ماشین و اسمبلی

```
mov     es,ax
;*****
lea     dx,msg1
call    print
lea     dx,str1
call    read
;*****
mov     cl,len1
mov     ch,0 ;cx = len1
lea     si,str1_1
call    atoi
mov     n,ax
;*****
mov     di,1
w1:
cmp     di,n
jge     ew1
mov     ax,n
mov     dx,0
div     di
cmp     dx,0
jne     rep1
add     sum,di
rep1:inc di
jmp     w1
ew1:
mov     ax,n
cmp     ax,sum
je      comp
lea     dx,str3
call    print
jmp     111
comp:
lea     dx, str2
call    print
111:
lea     dx,pkey
mov     ah,9h
int     21h
mov     ah,1h
int     21h
mov     ax,4C00h
int     21h
ends
print  proc near
mov     ah,9h
```

زبان ماشین و اسمبلی

```
int      21h
ret
print   endp
read    proc near
        mov     ah,0Ah
        int     21h
        ret
read    endp
atoi   proc near
        mov     ax,0
        fatoi:
        jcxz    efatoi
        mov     bl,[si]
        sub     bl,48
        mov     bh,0
        mul     ten
        add     ax,bx
        inc     si
        loop    fatoi
        efatoi:
        ret
atoi   endp
itoa    proc near
        mov     cx,0
        witoa:
        cmp     ax,0
        je      ewito
        mov     dx,0
        div     ten
        add     dx,48
        push    dx
        inc     cx
        jmp     witoa
        ewito:
        fitoa:  jcxz    efitoa
        pop     dx
        mov     [si],dl
        inc     si
        loop    fitoa
        efitoa:
        mov     [si],'$'
        ret
itoa    endp
end start ; set entry point and stop the assembler.
```

زبان ماشین و اسمبلی

3-مثال: برنامه ای بنویسید که یک عدد و یک مبنا را دریافت کند و عدد را به مبنای مورد نظر برده و نشان دهد.

```
data segment
msg1 db 10,13,"please enter a digit:$"
s1 label byte
max1 db 6
len1 db ?
str1 db 6 dup('$')
msg2 db 10,13,"convert is:$"
msg3 db 10,13,"please enter a base number:$"
sign dw 0
n dw 0
ten dw 10
r dw 0
pkey db 10,13,"press any key...$"
ends
stack segment
dw 128 dup(0)
ends
code segment
start:
; set segment registers:
mov ax, data
mov ds, ax
mov es, ax
; print msg1
lea dx,msg1
mov ah,9
int 21h
; read string s1=number
lea dx,s1
mov ah,0ah
int 21h
;*****convert string to number*****
mov cl,len1
mov ch,0 ;cx=len1
lea si,str1
call atoi
mov n,ax
lea dx,msg3
mov ah,9
int 21h
lea dx,s1
mov ah,0ah
int 21h
```

زبان ماشین و اسمبلی

```
,*****convert base string to number *****
lea si,str1
mov cl,len1
mov ch,0
call atoi
mov r,ax
,*****
lea dx,msg2
mov ah,9
int 21h
mov ax,n
,*****convert digit to base r****
mov cx,0
w1: cmp ax,0
    je ew1
    mov dx,0
    div r
    add dx,48
    cmp dx,58
    jb l1
    add dx,7
l1:  push dx
    inc cx
    jmp w1
ew1:
f2:jcxz ef2
    pop dx
    mov ah,2
    int 21h
loop f2
ef2:
lea dx, pkey
mov ah, 9
int 21h ; output string at ds:dx
; wait for any key....
mov ah, 1
int 21h
mov ax, 4c00h ; exit to operating system.
int 21h
ends
atoi proc near
mov ax,0
cmp [si], '+'
je m1
cmp [si], '-'
```

زبان ماشین و اسمبلی

```
jne m3
mov sign,1
m1: inc si
    dec cx
m3:
f1: jcxz ef1
    mov bl,[si]
    sub bl,48
    mul ten
    mov bh,0
    add ax,bx
    inc si
    loop f1
ef1:
cmp sign,1
jne m2
neg ax
m2:mov sign,0
ret
atoi endp
end start ; set entry point and stop the assembler.
```

4-مثال برنامه ای بنویسید که یک عدد را گرفته تمامی اعداد کامل کوچکتر از آن را چاپ کند.

```
data segment
msg1 db 10,13,"please enter a digit:$"
s1 label byte
max1 db 6
len1 db ?
str1 db 6 dup('$')
str2 db 6 dup('$')
ten dw 10
n dw 0
temp dw 0
sum dw 0
pkey db 10,13,"press any key...$"
ends
```

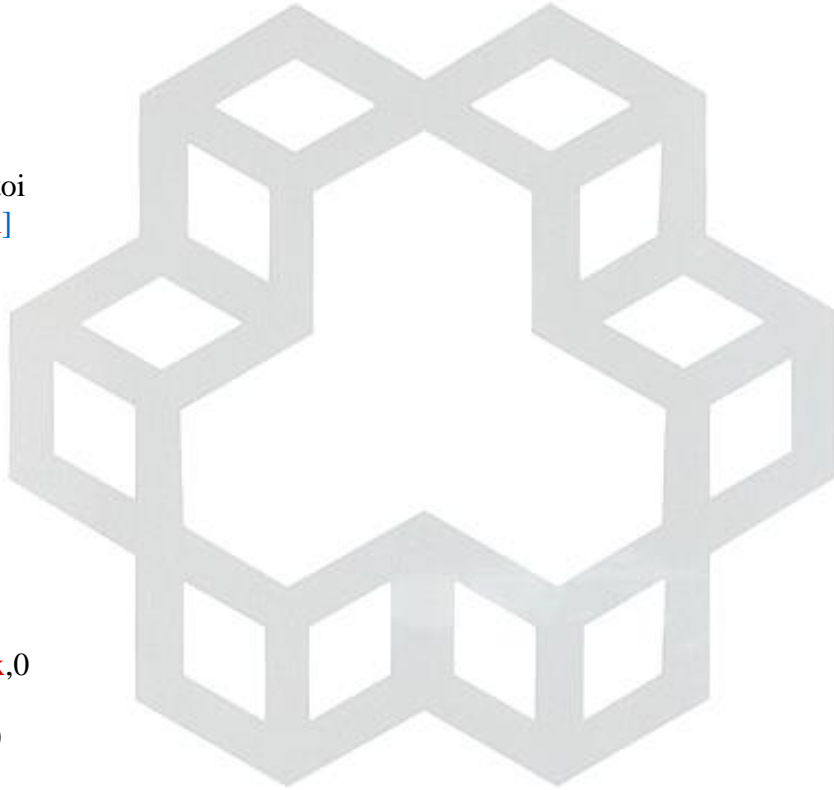
```
stack segment
dw 128 dup(0)
ends
code segment
start:
; set segment registers:
mov ax, data
mov ds, ax
mov es, ax
```

زبان ماشین و اسمبلی

```
; add your code here
lea dx,msg1
call print
lea dx,s1
call read
lea si,str1
mov cl,len1
mov ch,0 ;cx=le1
call atoi
mov n,ax
mov di,2
w2: cmp di,n
    jge ew2
mov sum,0
mov bx,1
mov temp,di
shr temp,1
w1:cmp bx,temp
    jg ew1
    mov ax,di
    mov dx,0
    div bx
    cmp dx,0
    jne l1
    add sum,bx
l1:inc bx
    jmp w1
ew1:
    mov ax,di
    cmp sum,ax
    jne l2
    lea si,str2
    call itoa
    lea dx,str2
    call print
l2: inc di
    jmp w2
ew2: lea dx, pkey
    mov ah, 9
    int 21h ; output string at ds:dx
    ; wait for any key....
    mov ah, 1
    int 21h
    mov ax, 4c00h ; exit to operating system.
    int 21h
```

زبان ماشین و اسمبلی

```
ends
print proc near
    mov ah,9
    int 21h
    ret
print endp
read proc near
    mov ah,0ah
    int 21h
    ret
read endp
atoi proc near
    mov ax,0
    fatoi:jcxz efatoi
        mov bl,[si]
        sub bl,48
        mov bh,0
        mul ten
        add ax,bx
        inc si
    loop fatoi
    efatoi:
    ret
atoi endp
itoa proc near
    mov cx,0
    witoa: cmp ax,0
        je ewitoa
        mov dx,0
        div ten
        add dx,48
        push dx
        inc cx
    jmp witoa
    ewitoa:
    fa:jcxz efa
        pop dx
        mov [si],dl
        inc si
    loop fa
    efa: mov [si],' '
        inc si
        mov [si],'$'
        ret
    itoa endp
```



دانشگاه صنعتی خواجه نصیرالدین طوسی

زبان ماشین و اسمبلی

end start ; set entry point and stop the assembler.

ضمیمه



منابع مطالعه بیشتر

1. زبان کامپیوترهای شخصی IBM و برنامه نویسی ماشین - پیترا بیل
2. برنامه نویسی اسمبلی در کامپیوترهای سازگار با - X86 مزیدی
3. مرجع کامل برنامه نویسی به زبان اسمبلی از 8086 تا پنتیوم. مهندس عین الله جعفر نژاد قمی

ارتباط

saeidi_reza@yahoo.com



دانشگاه صنعتی خواجه نصیرالدین طوسی