فصل ۱. سیستم عددی چیست؟

راه‌های متفاوتی جهت نمایش دادن ارزش اعداد وجود دارد. مدت‌ها پیش بشـر آموخت که با اسـتفاده از چوب شـمارش را انجام دهد و بعدها یاد گرفت چطور تصـویر چوب را بر روی زمین و بالاخره بر روی کاغذ ترسـیم کند. برای مثال عدد پنج را به‌صـورت | | | | | (پنج‌خط عمودی) نمایش می‌داد.

بعد از آن رومی‌ها از نمادهای متفاوتی برای نمایش اعداد اسـتفاده کردند. به‌عنوان مثال نمایش | | | مانند گذشته عدد ۳ را نمایش می‌داد، اما نماد V برای نمایش عدد پنج و نماد X برای نمایش ارزش ۱۰ به کار رفت.

اطلاعاتی که در حافظه کامپیوتر ذخیره می‌شـود ترکیبی از نمادهای A-Z و a-z و 0-9 و علائمی نظیر #، {}، (، ؟، !، ... اسـت که به روش خاصی در حافظه کامپیوتر ذخیره می‌شود. به‌عبارت‌دیگر می‌توان اطلاعاتی را که در کامپیوتر ذخیره می‌شود و به دو گروه اصلی تقسیم کرد:

۱. داده‌های رشته‌ای (کاراکتری)
۲. داده‌های عددی
   ۲-۱- اعداد اعشاری
   ۲-۲- اعداد صحیح

در سـیستم‌های عددی، هر رقم از عدد برحسب موقعیت مکانی آن دارای ارزش معینی است. در چنین سیستمی می‌توان برحسب موقعیت مکانی، اعداد را نمایش داد:

$$N = (a_{n-1}\, a_{n-2}\, \ldots\, a_2\, a_1\, a_0\, a_{-1}\, a_{-2}\, a_{-3}\, \ldots\, a_{-m})_B$$

$N = a_{n-1} * B^{n-1} + a_{n-2} * B^{n-2} + \ldots + a_2 * B^2 + a_1 * B^1 + a_0 * B^0 + a_{-1} * B^{-1} + a_{-m} * B^{-m}$

$$N = \sum_{k=-m}^{n-1} (a_k B^k)$$

B: مبنای سیستم عددی(base) n: تعداد رقم صحیح عدد

m: تعداد رقم اعشاری a_{n-1}، ... a_2، a_1، a_0، a_{-1}، ...، a_{-m}: ضرایب عدد

– مثال:

$$(754.05)_{10} = \sum_{k=-2}^{2} a_k 10^k = a_{-2} * 10^{-2} + a_{-1} * 10^{-1} + a_0 * 10^0 + a_1 * 10^1 + a_2 * 10^2$$
$$= 5 * 10^{-2} + 0 * 10^{-1} + 4 * 10^0 + 5 * 10^1 + 7 * 10^2$$
$$= \frac{5}{100} + 0 + 4 + 50 + 700$$

همان‌طور که در بالا مشـخص است در سیستم ده‌دهی[۱] از نمادهای ۹–۰ جهت نمایش اعداد استفاده می‌شود و مبنای این سیستم عدد ۱۰ است.

۱–۱. سیستم اعداد دودویی

همان‌طور که می‌دانید حداقل تاکنون کامپیوترها به‌اندازه انسان‌ها هوشمند نشده‌اند؛ بنابراین ساختن یک ماشین الکترونیکی که تنها بتواند دو حالت روشن (1) و یا خاموش (0) را نمایش دهد، کار آسان‌تری است؛ بنابراین کامپیوترها از سیستم دودویی استفاده می‌کنند. سیستم دودویی[۲] از دو رقم ۱ و ۰ در مبنای ۲ استفاده می‌کند. هر رقم در سیستم دودویی یک بیت نامیده می‌شود. هر ٤ بیت را یک نیبِل[۳]، هر ۸ بیت را یک بایت[٤]، هر۲ بایت را یک کلمه[۵]، و هر دو کلمه را یک کلمه مضاعف[٦] می‌نامیم (شکل ۱–۱).

---

۱ Decimal system

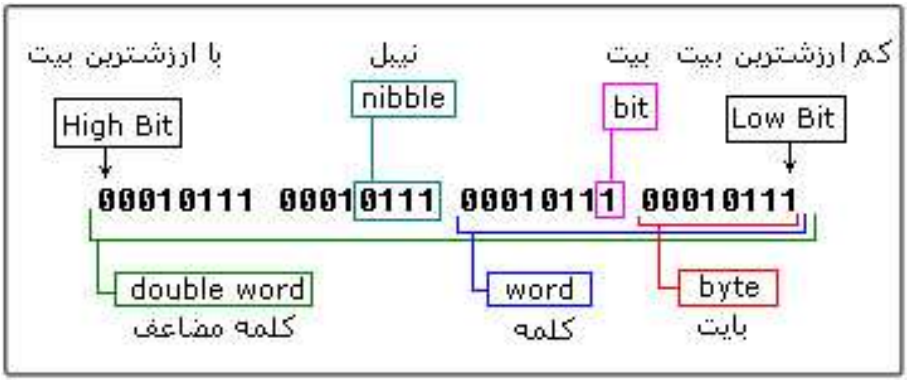
۲ Binary system

۳ nibble

٤ byte

۵ word

٦ double word



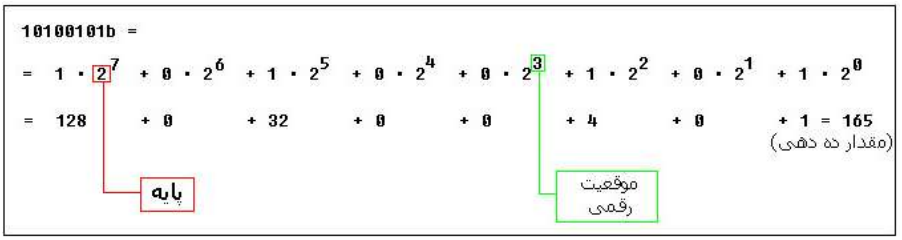
شکل ۱-۱) تفکیک و نام‌گذاری دسته بیت‌ها

به‌طور قراردادی در انتهای هر عدد دودویی علامت 'b' قرار می‌دهیم. با این روش می‌توانیم اعداد دودویی را از سایر مبناها تفکیک کنیم.

در این کتاب به‌طور مثال عدد 101b نمایشگر عدد پنج در مبنای ده‌دهی و عدد 101 نمایشگر عدد صد و یک در مبنای ده‌دهی است.

۱-۱-۱. تبدیل اعداد دودویی به مبنای ده‌دهی

عدد باینری 10100101b معادل عدد ۱۶۵ در مبنای ده است (شکل ۱-۲). همان‌طور که مشاهده می‌کنید روش کار به این صورت است که عدد ۲ (مبنا) را به توان موقعیت مکانی هر رقم باینری رسانده و در آن رقم (صفر یا یک) ضرب می‌کنیم.



شکل ۱-۲) روش تبدیل عدد دودویی به ده‌دهی

در مثال زیر روش تبدیل عدد اعشاری 11000.01b در مبنای دودویی به مبنای ده‌دهی و روش تبدیل عکس آن آورده شده است.

$$(11000.01)_2 = 1*2^4 + 1*2^3 + 0*2^2 + \ldots + 0*2^{-1} + 1*2^{-2}$$
$$= 16 + 8 + 0 + \ldots + 0 + ¼ = 24.25$$

برای تبدیل قسمت صحیح عدد مبنای ده به هر سیستم عددی دیگری، باقیمانده تقسیمات متوالی قسمت صحیح عدد ده‌دهی بر مبنای سیستم عددی مد نظر را گردآوری می‌کنیم (مراجعه شود به ۱-۲-۱).

$$24 = (11000)_2$$

و جهت تبدیل قسمت اعشاری عدد مبنای ده به سیستم عددی مد نظر (در اینجا سیستم عددی دودویی)، از روش ضرب متوالی در مبنای آن سیستم عددی استفاده می‌شود با این توضیح که در هر بار ضرب قسمت صحیح حاصل‌ضرب را به‌عنوان نتیجه آن مرحله نگهداری کرده و قسمت اعشاری را به مرحله بعد می‌بریم. به‌عنوان مثال:

$$0.25 * 2 = 0.5 \longrightarrow 0.5 * 2 = 1.0$$

عمل ضرب متوالی تا زمانی که قسمت اعشاری صفر شود و یا اینکه به تعداد رقم یعد اعشار دلخواه برسیم ادامه خواهد یافت. در نهایت با ترکیب دو بخش صحیح و اعشاری، کل عدد از مبنای ده به سیستم عددی تبدیل خواهد شد:

$$(24.25)_{10} = (11000.01)_2$$

۱-۲. سیستم اعداد شانزده‌شانزدهی[۷]

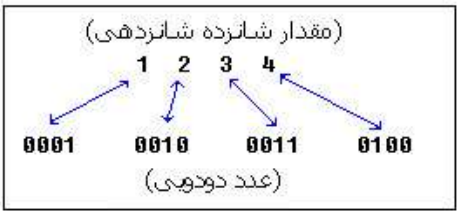
به دلیل اینکه خواندن اعداد در سیستم شانزده‌شانزدهی مختصر و آسان است (جدول ۱-۱) و همچنین تبدیل اعداد دودویی به این سیستم و برعکس، نیز بسیار راحت است (شکل ۱-۳)، بنابراین اغلب از سیستم شانزده‌شانزدهی جهت نمایش اعداد استفاده می‌کنیم. در سیستم شانزده‌شانزدهی نمادهای زیر جهت نمایش ارقام صفر تا پانزده زیر استفاده می‌شود:

0-1-2-3-4-5-6-7-8-9-A-B-C-D-E-F

[۷] hexadecimal

جدول ۱-۱) جدول معادل‌سازی اعداد در مبنای ده، دو و شانزده

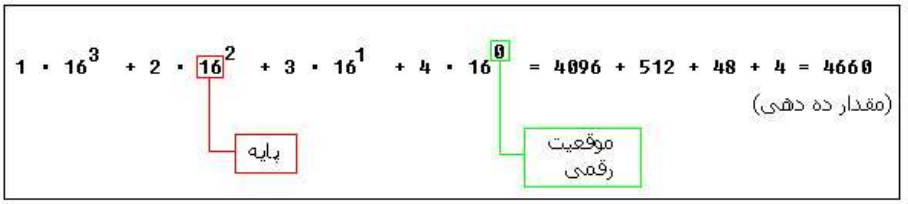
مبنای ده	مبنای دو	مبنای شانزده
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F



شکل ۱-۳) تبدیل اعداد مبنای ۱۶و ۲

در این سیستم به‌طور قراردادی در انتهای اعداد علامت 'h' قرار می‌گیرد و هر ٤ بیت (نیبِل) را می‌توان با یک رقم نمایش داد (شکل ۱-۳). اگر اعداد شانزده‌شانزدهی با یکی از حروف F,...,B,A آغاز شوند، در ابتدای عدد صفر قرار داده می‌شود. به‌عنوان مثال: 0E15h

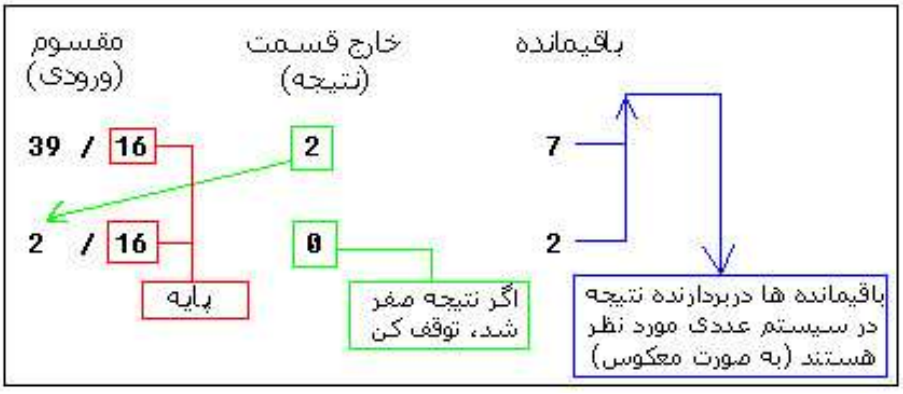
همچنین جهت تبدیل اعداد شانزده‌شانزدهی به معادل ده‌دهی آن‌ها، از روشی مشابه با تبدیل اعداد دودویی به ده‌دهی (بخش ۱-۱-۱) استفاده می‌کنیم. به این صورت است که عدد ۱۶ (مبنا) را به توان موقعیت مکانی هر رقم شانزده‌شانزدهی رسانده و سپس در همان رقم ضرب می‌کنیم. عدد 1234h در مبنای شانزده‌شانزدهی معادل ٤٦٦٠ در مبنای ده‌دهی است (شکل ۱-٤).



شکل ۱-٤) تبدیل عدد از مبنای شانزده به مبنای ده

۱-۲-۱. تبدیل اعداد از سیستم ده‌دهی به شانزده‌شانزدهی

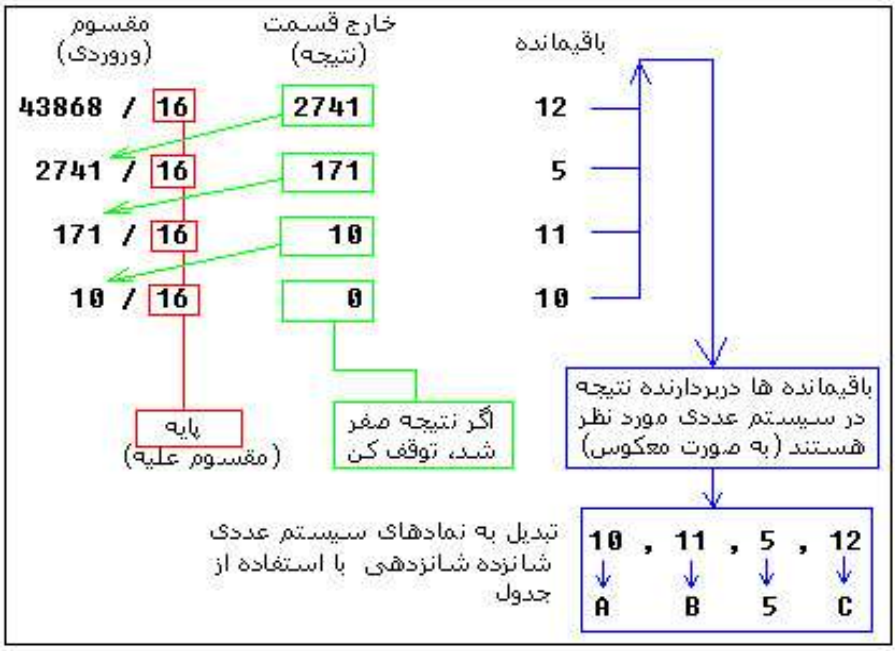
در اینجا نیز همانند تبدیل اعداد ده‌دهی به دودویی، از تقسیمات متوالی بر مبنای موردنظر استفاده می‌کنیم. به این مفهوم که هر دفعه باید خارج‌قسمت را حفظ و باقیمانده را نگهداریم. عمل تقسیم را تا زمانی که نتیجه صفر شود ادامه می‌دهیم، سپس باقیمانده‌ها (از راست به چپ) نتیجه را نمایش می‌دهند. به‌عنوان مثال اگر بخواهیم عدد ۱۰(۳۹) را به معادل شانزده‌شانزدهی آن تبدیل نماییم مانند شکل ۱-۵ عمل می‌کنیم: $(39)_{10}=(27)_{16}$



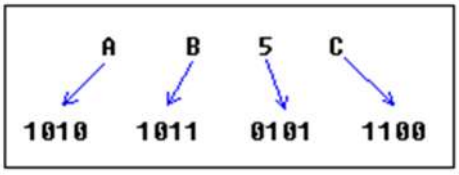
شکل ۱-۵) تبدیل عدد از مبنای ده به مبنای شانزده

اجازه دهید یک مثال با پیچیدگی کمی بیشتر را ببینیم: شکل ۱-٦ تبدیل عدد ده‌دهی ٤٣٨٦٨ به مبنای شانزده و سپس تبدیل آن به مبنای دودویی را نشان می‌دهد. همانطور که مشخص است جواب تبدیل عدد ٤٣٨٦٨ در مبنای ده به مبنای شانزده عدد 0AB5Ch است.

برای تبدیل این عدد از مبنای شانزده به مبنای دودویی می‌توانیم همان‌طور که قبلاً دیدیم از طریق جدول تبدیل (جدول ۱-۱) معادل دودویی هر یک از ارقام را جایگزین نماییم.همان‌طور که در شکل ۱-۷ مشاهده می‌کنید حاصل عدد AB5C در مبنای شانزده‌شانزدهی، عدد 1010101101011100b در مبنای دودویی است.



شکل ۱-٦) تبدیل عدد از مبنای ۱۰ به مبنای ۱٦



شکل ۱-۷) تبدیل مبنای ۱٦ به مبنای دودویی

۱-۳. روش نگهداری اعداد در کامپیوتر

در یک ماشین الکترونیکی، سیستم اعداد انتخاب‌شده برای نگهداری اطلاعات باید قابلیت نگهداری اعداد بدون علامت و اعداد علامت‌دار را داشته باشد.

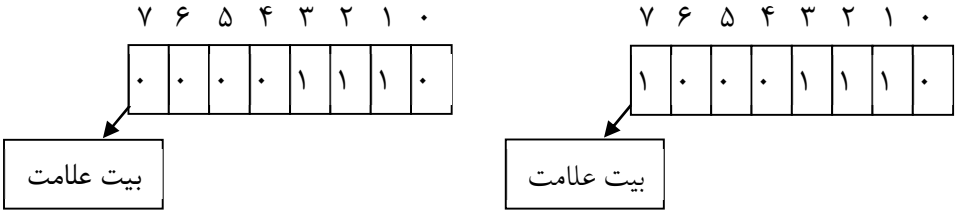
سه روش برای این کار وجود دارد:

۱-روش علامت مقدار ۲-روش مکمل ۱ ۳-روش مکمل ۲

در سیستم‌های کامپیوتری از روش سوم برای نگهداری اعداد استفاده می‌شود.

در روش علامت مقدار بالاترین ارزش بیت موجود در کلمه به‌عنوان نمایش علامت به کار می‌رود. به این مفهوم که اگر باارزش‌ترین بیت مقدار صفر داشته باشد، عدد نگهداری شده در کلمه «مثبت» تعبیر می‌شود و درصورتی‌که یک باشد عدد موجود در کلمه «منفی» تعبیر می‌شود. به‌طور مثال:

$(-۱٤)_{۱۰} = -(۱۱۱۰)_۲$ $(۱٤)_{۱۰} = (۱۱۱۰)_۲$



در روش علامت مقدار بازه عددی نمایش داده‌شده توسط یک بایت (اعداد علامت‌دار) 128- الی 127 است و بازه عددی اعداد مثبت 0 تا 255 است.

هیچ روشی وجود ندارد که نشان دهد به‌طور قطع، بایت شانزده‌شانزدهی 0FFh مثبت است یا منفی، به‌عبارت‌دیگر این بایت می‌تواند نمایشگر هر یک از اعداد ده‌دهی ۲۵۵ و یا ۱- باشد. برای آشنا شدن با روش مکمل-۲ باید ابتدا با روش مکمل-۱ آشنا شویم.

۱-۳-۱. روش مکمل-۱ (R-1)

برای به دست آوردن مکمل R-1 (R مبنای سیستم عددی است) باید هر رقم از عدد موردنظر را از مقدار R-1 کم کنیم.

به‌عنوان مثال برای به دست آوردن مکمل-۹ عدد دهدهی ۸۵، به این صورت عمل می‌کنیم:

$(14)_{10}$ = مکمل-۹ عدد $(85)_{10}$ ⟵ ۹۹-۸۵=۱۴ ⟵ R-۱=۹

$(10)_{2}$ = مکمل-۱ عدد $(1101)_{2}$ ⟵ ۱۱۱۱-۱۱۰۱=۰۰۱۰ ⟵ R-۱=۱

۱-۳-۲. روش مکمل-۲ (R)

در این روش برای یافتن مکمل R یک عدد، ابتدا باید مکمل R-1 عدد را پیداکرده، سپس به مقدار به‌دست‌آمده، عدد یک را اضافه نمایید.

۱۵ = ۱+ ۱۴= مکمل ۱۰ عدد $(85)_{10}$

۰۰۱۱ = ۱+ ۰۰۱۰ = مکمل ۲ عدد $(1101)_{2}$

نکته: در مبنای ۲، دو روش برای به دست آوردن مکمل-۲ یک عدد وجود دارد.

روش اول: در این روش ابتدا باید تمام صفرها را به یک و همه یک‌ها را به صفر تبدیل کنیم، سپس حاصل را با ۱ جمع می‌کنیم.

روش دوم: در این روش که به روش سریع معروف است، برای تبدیل یک عدد دودویی به مکمل-۲ آن سه مرحله‌ی زیر را به‌ترتیب انجام می‌دهیم:

۱- از سمت راست عدد شروع کرده تا زمانی که به یک نرسیده‌ایم تمام بیت‌ها را به خروجی می‌بریم.

۲- اولین یک را نیز به خروجی منتقل می‌کنیم.

۳- بقیه بیت‌ها را نقیض (تبدیل صفر به یک و برعکس) می‌کنیم.

نکته: در هنگام تبدیل یک عدد دودویی به مکمل آن، آنچه مهم است و باید در ابتدا مشخص شود، تعداد بیت‌هایی است که برای نمایش عدد در نظر گرفته‌شده است. به‌عبارت‌دیگر قبل از محاسبه مکمل-۲ یک عدد باید به صفرهای بی‌اهمیت عدد نیز توجه داشت.

۱۱۰۱ ⟵ ۰۰۱۱ (تبدیل مکمل-۲ عدد ۱۱۰۱ بدون صفر بی‌اهمیت)

۰۱۱۰۱ ⟵ ۱۰۰۱۱ (تبدیل مکمل-۲ عدد ۱۱۰۱ با یک صفر بی‌اهمیت)

۰۰۱۱۰۱ ⟵ ۱۱۰۰۱۱ (تبدیل مکمل-۲ عدد ۱۱۰۱ با دو صفر بی‌اهمیت)

۰۰۰۰۱۱۰۱ ⟵ ۱۱۱۱۰۰۱۱ (تبدیل مکمل-۲ عدد ۱۱۰۱ با چهار بی‌اهمیت)

۱-٤. آشنایی با ابزارهای شبیه‌ساز emu8086

ابزارهای دستی بسیاری در emu8086 برای تبدیل اعداد و انجام محاسبات عبارات عددی وجود دارد، برای استفاده از این ابزارها، باید روی منو math کلیک کنیم.

۱-٤-۱. تبدیل کننده پایه[۸]

به شما اجازه می‌دهد که اعداد از هر سیستمی را به سیستمی دیگر تبدیل نمایید. تنها کار لازم این است که مقدار موردنظر خود را در یکی از محل‌های ورودی متن قرار داده تا به‌صورت خودکار به سایر سیستم‌های عددی تبدیل شود. در این ابزار می‌توان با مقادیر ۸ بیتی و یا ۱٦ بیتی کارکنیم.

شکل ۱-۸) پنجره تبدیل کننده پایه

[۸] Base converter

۱-٤-۲. محاسبه‌گر چندمبنایی[۹]

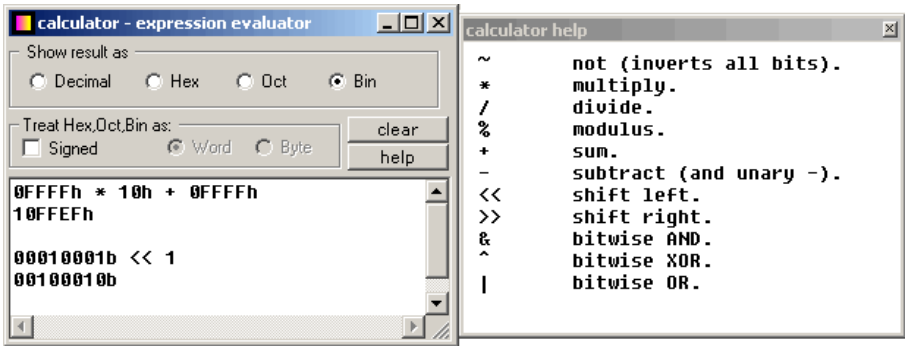
توسط این ابزار محاسبات بین اعداد در سیستم‌های عددی متفاوت و تبدیل اعداد از یک سیستم عددی به سیستم دیگر را می‌توان انجام داد. برای این کار کافی است عبارت‌های محاسباتی را تایپ و سپس دکمه enter را فشار دهیم، سپس نتیجه در سیستم عددی انتخاب‌شده نمایش داده خواهد شد.

می‌توان محاسبات حداکثر تا ۳۲ بیت را توسط این ابزار انجام داد. زمانی که دکمه signed انتخاب شود (تیک بخورد) محاسبه‌گر عبارات فرض می‌کند که تمام ارزش‌ها به‌صورت علامت‌دار هستند (به‌جز اعداد ده‌دهی و کلمات مضاعف).

کلمه مضاعف همواره به‌صورت مقادیر علامت‌دار رفتار می‌کند، به‌طور مثال عدد 0FFFFFFFFh به عدد ۱- تبدیل می‌شود.

برای مثال اگر بخواهید مقادیر 0FFFFh * 10h + 0FFFFh (بالاترین موقعیت حافظه که می‌توان در پردازشگر ۸۰۸٦ به آن دسترسی داشت) را محاسبه کنیم و خانه‌های signed و word را از قبل انتخاب کرده باشیم (تیک بزنیم) حاصل ۱۷- خواهد شد (زیرا عبارت به‌صورت (1-) + 16 * 1- محاسبه می‌شود)

برای محاسبه مقادیر بدون علامت، باید تیک signed را برداشته تا عبارت 65535 * 65535 + 16 محاسبه شود (حاصل 1114095 می‌شود).



شکل ۱-۹) پنجره «محاسبه‌گر چندمبنایی» یا «ارزیاب عبارت»

[۹] Multi base calculator

شـما همچنین می‌توانید از تبدیل‌کننده پایه برای تبدیل اعداد غیر ده‌دهی به اعداد ده‌دهی علامت‌دار اسـتفاده کنید و محاسـبات را بر اسـاس ارقام ده‌دهی انجام دهید (اگر آسـان‌تر است). عملگرهای جدول ۱-۲ توسط محاسبه‌گر پشتیبانی می‌شوند.

جدول ۱-۲) عملگرهای پشتیبانی شده توسط محاسبه‌گر چندمبنایی

عملگر	توصیف عملگر		عبارت محاسبتی	نتیجه
~	Not (inverts all bits)	نقیض کلیه بیت‌ها	~101011b	**00010100 b**
*	Multiply	ضرب	1010b * 100b	**00101000 b**
/	Divide	تقسیم	1010b / 10b	**00000101 b**
%	Modulus	باقیمانده	1010b % 101b	**00000010 b**
+	Sum	جمع	1010b + 101b	**00001111 b**
-	Subtract (and unary)	تفریق و منفی	1010b - 101b	**00000101 b**
<<	Shift left	تغییر مکان به چپ	100b << 2	**00010000 b**
>>	Shift right	تغییر مکان به راست	100b >> 2	**00000001 b**
&	Bitwise AND	بیت‌های مشابه AND	1010b & 0011b	**00000010 b**
^	Bitwise XOR	بیت‌های مشابه XOR	1010b ^ 0011b	**00001001 b**
|	Bitwise OR	بیت‌های مشابه OR	1010b | 0011b	**00001011 b**

یادآوری:

جهت محاسبات در مبنای-۲ باید در انتهای عدد کاراکتر 'b' نوشته شود.

به‌عنوان مثال: 01011110b

به انتهای اعداد شـانزده‌شـانزدهی باید کاراکتر 'h' اضـافه شـود و زمانی که عدد با یکی از حروف A-F آغاز شود باید در ابتدای عدد 0 (صفر) بگذارید.

به‌عنوان مثال: 0ABCDh

همچنین برای نوشتن اعداد در مبنای-۸ (octal) باید در انتهای عدد کاراکتر 'o' بگذاریم.

به‌عنوان مثال: 77O

۱-۵. تمرین فصل یک

۱. هر بایت از ... بیت و هر کیلوبایت از ... بایت تشکیل‌شده است.

۲. هر بایت ... و هر کلمه ... وضعیت مختلف از صفر و یک را می‌تواند نمایش دهد. به‌عبارت‌دیگر در هر بایت اعداد صفر تا ... + و یا ... – تا ... + را می‌توان قرار داد.

۳. اعداد 21، 21- و 38.75 در سیستم ده‌دهی را به دودویی تبدیل نمایید.

٤. اعداد 11010101 و 1010101.1011 را به سیستم ده‌دهی و شانزده‌شانزدهی تبدیل نمایید.

۵. اعداد $(9FBA)_{16}$ و $(7245)_8$ را به سیستم دودویی تبدیل نمایید.

٦. عملیات زیر را ابتدا به‌صورت دستی انجام و سپس توسط ابزار «محاسبه‌گر چندمبنایی» شبیه‌ساز emu8086 (بخش ۱-٤-۲) صحت عملیات خود را بررسی نمایید:

$10111110+$	$24AD+$	$10110101-$	$2BA8-$
10101111	$F2A$	10001011	$17EF$

۷. با استفاده از ابزار «تبدیل‌کننده پایه» شبیه‌ساز (بخش ۱-٤-۱) مقدار دودویی کاراکتر 'a' و 'A' را به دست آورده و سپس توسط ابزار «محاسبه‌گر چندمبنایی» شبیه‌ساز (بخش ۱-٤-۲) تفاوت آن‌ها را در مبنای دو محاسبه و معادل ده‌دهی آن را به دست آورید.

۸. حاصل عبارت محاسباتی 14-25 را با روش مکمل ده محاسبه نمایید.

۹. مکمل دو نیبل 1001 و بایت1001 را محاسبه نمایید.

فصل ۲. زبان اسمبلی چیست؟

این خودآموز برای افرادی است که هیچ آشنایی با زبان اسمبلی ندارند یا آشنایی آن‌ها با این زبان بسیار کم است. البته اگر اطلاعاتی در مورد زبان‌های برنامه‌نویسی سطح بالا مانند جاوا، بیسیک، c، و یا c++ داشته باشید، دانش برنامه‌نویسی‌تان کمک زیادی به شما خواهد کرد. اگر شما آشنا با زبان اسمبلی باشید، استفاده از این خودآموز در آشنایی با نحوه‌ی برنامه‌نویسی با شبیه‌ساز emu8086 کمک شایانی خواهد کرد.

یادآوری: در این کتاب جهت اختصار به‌جای «شبیه‌ساز emu8086» تنها کلمه «شبیه‌ساز» آورده شده است.

۲-۱. آشنایی مختصر با معماری کامپیوتر

همان‌طور که می‌دانید، پردازنده[۱] قلب هر سیستم است، به‌طوری‌که کامپیوترها را اغلب با نام پردازنده آن‌ها می‌شناسند. پردازنده‌ای که در اینجا با آن‌ها سروکار داریم، طراحی و ساخت شرکت اینتل[۲] هستند. شرکت اینتل تاکنون پردازنده‌های مختلفی را به بازار عرضه کرده است، از جمله‌ای این پردازنده‌ها، ریزپردازنده ۸۰۸۶ (یا iAPX 86) است که یک تراشه ریزپردازنده ۱۶ بیتی است که از اوایل سال ۱۹۷۶ طراحی و در اواسط سال ۱۹۷۸ به بازار معرفی شد. چندی بعد در سال ۱۹۷۹، شرکت اینتل ریزپردازنده ۸۰۸۸ را برپایه تراشه

[۱] CPU: Central Processing Unit

[۲] Intel

۸۰۸۶ و با اصلاح مختصری (گذرگاه ۸ بیتی داده خارجی) به بازار عرضه کرد (شکل۲-۱). قابل‌توجه است که تراشه اینتل ۸۰۸۶ به‌عنوان پردازنده در طراحی اصلی IBM PC، از جمله نسخه گسترده‌ای به نام IBM PC XT مورد استفاده قرارگرفته است. به‌طور قطع می‌توان گفت که تراشه ۸۰۸۶ معماری x86 را به ارمغان آورد که در نهایت موفق‌ترین پردازنده اینتل شد.

وظیفه هر پردازنده را می‌توان به‌طور اختصار به شکل زیر بیان کرد:

۱. **واکشی دستور بعدی:** قرار دادن دستور بعدی و به‌روزرسانی شمارنده دستورات برنامه جاری
۲. **رمزگشایی دستور:** ترجمه آدرس و واکشی عملوندهای آن دستور از حافظه اصلی
۳. **اجرای دستور:** انجام محاسبات مورد نیاز مربوط به هر دستور و ذخیره نتیجه در حافظه و تغییر ثبات وضعیت (ثبات flag یا پرچم)

در شکل ۲-۱ «واحد کنترل»[۳] عملیات مربوط به دستورالعمل‌ها و داده‌ها را کنترل کرده، آدرس‌ها را برای «واحد محاسبه و منطق»[۴] رمزگشایی می‌کند.

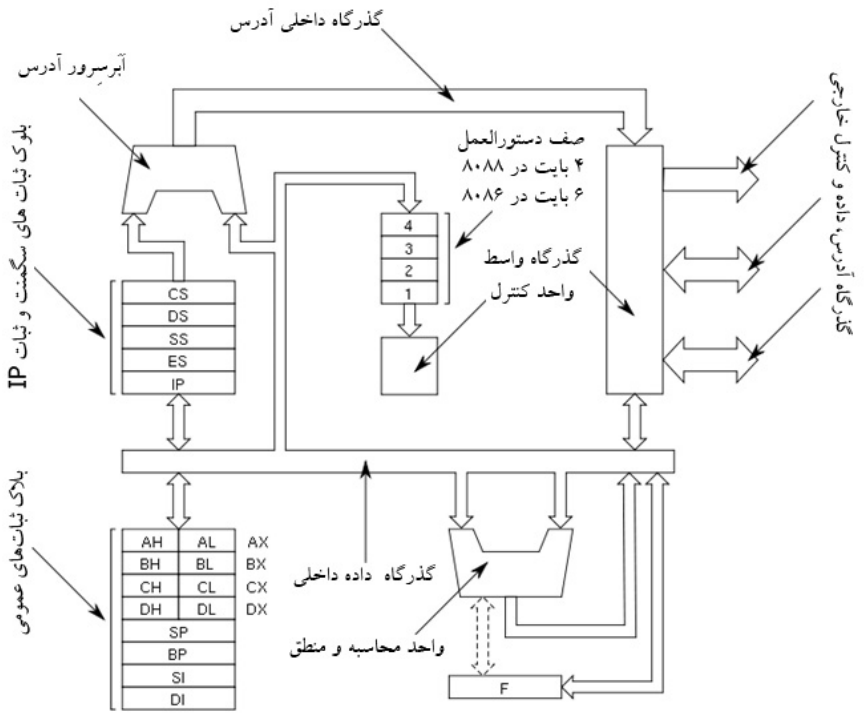
واحد محاسبه و منطق وظیفه انجام کلیه عملیات محاسباتی و منطقی را برعهده دارد. تمامی اعمال پردازنده توسط یک ساعت درونی با «حافظه اصلی»[۵] هماهنگ می‌شود.

---

[۳] CU: Control Unit

[۴] ALU: Arithmetic Logic Unit

[۵] RAM: Random-Access Memory



شکل ۲-۱) دیاگرام داخلی ریزپردازنده ۸۰۸۶ و ۸۰۸۸

۲-۲. گذرگاه داده و آدرس[۶]

گذرگاه داده داخلی پردازنده، مجموعه‌ای از سیم‌های موازی است که داده‌ها (الکترون‌ها) را بین بخش‌های مختلف پردازنده انتقال می‌دهد. وقتی داده‌ها از حافظه اصلی خوانده شدند، واحد کنترل آدرس آن‌ها را محاسبه کرده و آن‌ها را در گذرگاه آدرس قرار می‌دهد.

واحد حافظه که در داخل ریزپردازنده است، گذرگاه آدرس را می‌خواند و داده‌های درخواستی را در گذرگاه داده قرار می‌دهد.

همچنین سیگنالی به پردازنده فرستاده و اعلام می‌کند داده‌ها قابل برداشت از گذرگاه داده هستند. آنگاه پردازنده داده‌ها را از گذرگاه خارجی به ناحیه حافظه داخلی خود منتقل[۷]

---

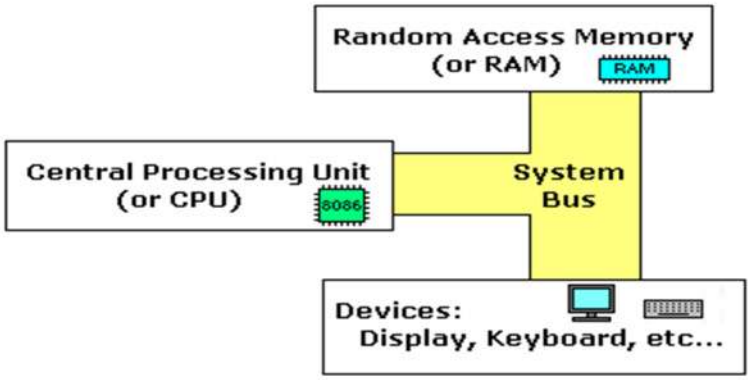
[۶] Address BUS & Data BUS

[۷] Cache

می‌کنـد. در اینجـا بـه تعریف عبـارات ســـاعـت پردازنـده و ثبـات، بـه دلیل اهمیت آن‌ها می‌پردازیم:

ساعت[۸]: هر عملی در پردازنده مرکزی باید توسـط یک ساعت داخلی همگام شود. اسـاسی‌ترین واحد مربوط به دستورات ماشین را چرخه ماشین یا چرخه ساعت می‌نامند که برحسب میلیون چرخه بر ثانیه و با واحد MHz مشخص می‌شود. ساعت اصلی ریزپردازنده ۸۰۸۶ بین 5 MHz الی 10 MHz اسـت، درحالی‌که ساعت اصلی اغلب پردازنده‌های پنتیوم بین 200MHz یا 400MHz است.

ثبات[۹]: حافظه‌ای بسـیار سریع است که مستقیماً با «واحد محاسبه و منطق» و «واحد کنترل» در ارتباط اسـت و اطلاعات را با سرعت بسیار زیادتر از حافظه‌های دیگر دریافت و بـازیـابی می‌کنـد. ثبـات‌های حافظه ۸، ۱۶ یا ۳۲ بیتی هســـتند. پردازنده‌های مرکزی دارای گذرگاه داده داخلی به پهنای دو برابر گذرگاه داده خارجی می‌باشند.



شکل ۲-۲) نمودار ساده معماری سیستم کامپیوتری- ارتباط بین ریزپردازنده، حافظه اصلی، دستگاه‌های خارجی، و گذرگاه سیستم

---

[۸] Clock

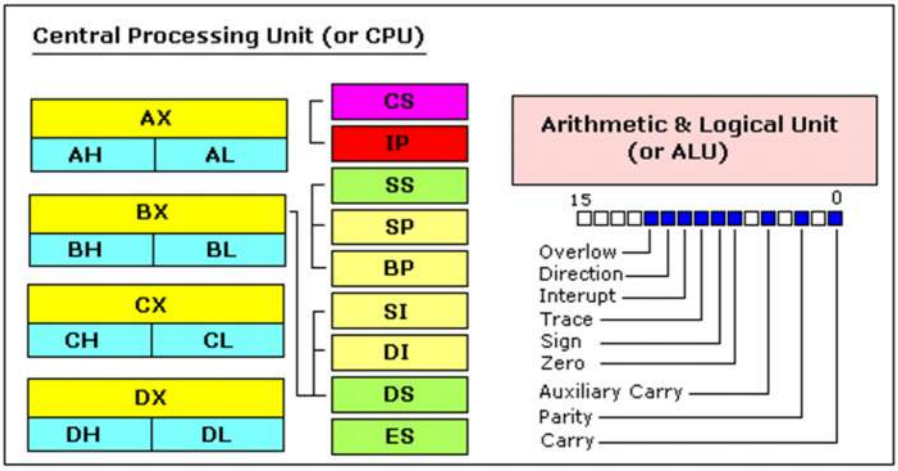
[۹] Register

مدل ارائه‌شده در شکل ۲-۲ یک مدل ساده از معماری یک کامپیوتر را نشان می‌دهد. در شکل فوق، گذرگاه سیستم (System Bus) قسمت‌های مختلف کامپیوتر را به یکدیگر مربوط کرده تا تبادل اطلاعات بین اجزای مختلف کامپیوتر وجود داشته باشد.

حافظه اصلی یا RAM جایی است که برنامه‌ها برای اجرا در آن‌ها بارگذاری می‌شوند. همان‌طور که قبلاً اشاره شد CPU قلب کامپیوتر است و اغلب محاسبات در داخل آن انجام می‌شود.

۲-۳. آشنایی با ثبات‌های پردازنده

کلیه پردازنده‌های اینتل دارای ٤ گروه ثبات شامل ثبات عمومی، ثبات اندیس (شاخص)، ثبات سگمنت و ثبات کنترل هستند. در پردازنده‌های 80386 به بعد، دو ثبات FS و GS علاوه بر ثبات‌های فوق جهت انجام امور خاص به CPU اضافه‌شده‌اند (شکل ۲-۳).



شکل ۲-۳) مجموعه ثبات‌های پردازنده ۸۰۸٦ (واحد پردازش مرکزی)

پردازنده 8086 هشت ثبات عمومی ١٦ بیتی دارد که هریک نام مخصوص به خود را دارند.

۲-۳-۱. ثبات‌های عمومی

AX[۱۰] : ثبات انباره که در اعمالی مورد استفاده قرار می‌گیرد که نیاز به ورودی و خروجی و محاسبات زیاد دارد. این ثبات به دو بخش AL(۸ بیت پایین) و AH(۸ بیت بالا) قابل‌تقسیم و آدرس‌دهی است.

BX[۱۱] : ثبات آدرس پایه که به دو بخش BL و BH قابل تقسیم و آدرس‌دهی است.

CX[۱۲] : ثبات شمارنده که به دو بخش CL و CH قابل آدرس‌دهی و بخش‌پذیری است.

DX[۱۳] : ثبات داده که به دو ثبات DH و DL تقسیم می‌شود.

SI[۱۴] : این ثبات به ثبات شاخص منبع معروف است.

DI[۱۵] : ثبات شاخص مقصد

BP[۱۶] : ثبات اشاره‌گر پایه

SP[۱۷] : ثبات اشاره‌گر پشته

علی‌رغم اسم در نظر گرفته‌شده برای هریک از ثبات‌ها، این برنامه‌نویس است که تعیین می‌کند از هر ثباتی چه استفاده‌ای خواهد شد. هدف اصلی یک ثبات، نگهداری یک مقدار (متغیر) است. در ۸۰۸۶ اندازه تمامی ثبات‌ها ۱۶ بیت است، به‌طوری‌که مقدار ۱۲۳۴۵ در مبنای ده‌دهی و یا معادل آن 0011000000111001b در مبنای دودویی در این ثبات‌ها جای می‌گیرد.

چهار ثبات عمومی AX،BX،CX و DX از دو بخش ۸ بیتی ساخته‌شده‌اند. برای مثال اگر AX=0011000000111111b آنگاه AH=00110000b و AL=00111111b خواهد بود؛

---

[۱۰] Accumulator Register

[۱۱] Base Address Register

[۱۲] Count Register

[۱۳] Data Register

[۱۴] Source Index Register

[۱۵] Destination Index Register

[۱۶] Base Pointer Register

[۱۷] Stack Pointer Register

بنابراین زمانی که هریک از ۸ بیت ثبات ۱۶ بیتی را بروز رسانی می‌کنیم محتوای ثبات ۱۶ بیتی، تغییر خواهد کرد و برعکس. این روند برای ۳ ثبات دیگر نیز دقیقاً به همین صورت انجام خواهد شد. در اینجا از کاراکتر 'H' برای نشان دادن ۸ بیت بالایی[۱۸] ثبات‌ها و از کاراکتر 'L' برای نمایش ۸ بیت پایینی[۱۹] ثبات‌ها، استفاده شده است.

به دلیل اینکه ثبات‌ها در داخل ریزپردازنده قرارگرفته‌اند، خیلی سریع‌تر از حافظه‌های دیگر عمل می‌کنند. دسترسی به یک مکان از حافظه اصلی (RAM) نیاز به استفاده از گذرگاه سیستم دارد؛ بنابراین شما باید سعی کنید متغیرهای خود را تا حد امکان در ثبات‌ها نگهداری کنید، تا بدون صرف زمان به محتوای آن‌ها دسترسی داشته باشید. مجموعه این ثبات‌ها شامل ظرفیت کمی هستند و در برخی از موارد ثبات‌های عمومی دارای اهداف خاص هستند که این موارد استفاده از آن‌ها را در غالب یک متغیر محدود می‌کند (در فصل چهارم یکی از این موارد را با بیان یک مثال مشاهده خواهید کرد)؛ اما با این وجود ثبات‌ها مکان‌های بسیار خوبی جهت ذخیره موقت داده‌ها به‌منظور محاسبات به شمار می‌روند.

۲-۳-۲. ثبات‌های سگمنت

CS[۲۰]: به بخشی[۲۱] از حافظه اصلی اشاره می‌کند که برنامه جاری در داخل آن قرار دارد. به‌طوری‌که اولین دستور اجرایی برنامه در ابتدا آن بخش قرار می‌گیرد. اگر برنامه بزرگ باشد (بیشتر از 64KB) می‌توان از چند سگمنت کد (قطعات دستورالعمل) استفاده کرد.

DS[۲۲]: به بخشی از حافظه اصلی اشاره می‌کند که داده‌ها و ناحیه کاری برنامه در آن قرار می‌گیرند. ممکن است یک برنامه، چندین ناحیه کاری داشته باشد که باید جداگانه تعریف شوند.

---

[۱۸] High

[۱۹] Low

[۲۰] Code Segment

[۲۱] segment

[۲۲] Data Segment

ES[۲۳]: همان‌طور که از اسم این ثبات مشخص است به یک بخش اضافی در حافظه اصلی اشاره می‌کند که برای تعریف داده‌های بیشتر و به‌خصوص برای انجام عملیات برروی رشته‌ها از آن استفاده می‌شود. اگر برنامه به بیش از یک ناحیه داده نیاز داشته باشد؛ می‌توانیم از طریق این ثبات به آن اشاره‌کنیم.

SS[۲۴]: به بخشی از حافظه اصلی که حاوی آدرس پشته است، اشاره می‌کند. به‌عبارتی‌دیگر این ثبات به سگمنت حاوی آدرس‌های برگشت، از یک زیر برنامه اشاره دارد؛ به‌طوری‌که هر نوع اطلاع لازم جهت فراخوانی زیر برنامه‌ها از طریق این ثبات قابل‌دستیابی است.

هرچند که ذخیره هر داده‌ای در ثبات‌های سگمنت امکان‌پذیر است اما انجام این عمل هرگز توصیه نمی‌شود. ثبات‌های سگمنت برای انجام اهداف خاصی (دسترسی به بلاک‌های حافظه اصلی) در نظر گرفته‌شده‌اند.

جهت دسترسی به هر قسمت از حافظه اصلی، ثبات‌های سگمنت با ثبات‌های عمومی ترکیب می‌شوند. برای مثال اگر بخواهیم به آدرس فیزیکی 12345h (در مبنای شانزده) دسترسی داشته باشیم، آنگاه باید DS=1230h و SI=0045h مقداردهی شود. این روش به ما این امکان را می‌دهد که دسترسی بیشتری به حافظه فیزیکی داشته باشیم به‌جای آنکه بخواهیم از یک ثبات تنها با محدودیت ۱۶ بیتی استفاده کنیم. ریزپردازنده جهت محاسبه آدرس فیزیکی، ثبات سگمنت را در 10h ضرب کرده و سپس با ثبات عمومی جمع می‌کند:

DS * 10h + SI = 1230h * 10h + 0045h = 12300h + 0045h = 12345h

آدرس به‌دست‌آمده از دو ثبات به شماره‌ی بایتی (یا کلمه‌ای) در حافظه اصلی اشاره می‌کند و به آن آدرس مؤثر[۲۵] گفته می‌شود.

به‌طور قراردادی ثبات‌های BX، SI، و DI با ثبات سگمنت DS کار می‌کنند و ثبات BP و SP با ثبات سگمنت SS کار می‌کند. سایر ثبات‌های عمومی امکان تشکیل یک آدرس مؤثر

[۲۳] Extra Segment

[۲۴] Stack Segment

[۲۵] effective address

را ندارند. همچنین گرچه BX می‌تواند یک آدرس مؤثر را شکل می‌دهد اما BH و BL امکان این کار را ندارند. فصل سوم این کتاب به‌طور جامع به این موضوع اختصاص یافته است.

۲-۳-۳. ثبات‌های خاص

دو ثبات زیر جهت انجام امور خاص توسط سیستم‌عامل به کار می‌روند:

ثبات IP[۲۶]: اشاره‌گر به دستورالعمل بعدی

ثبات پرچم[۲۷]: تعیین‌کننده حالت جاری ریزپردازنده

ثبات IP همیشه با ثبات سگمنت CS کار می‌کند و به دستورالعمل اجرایی بعدی اشاره می‌کند. ثبات پرچم (که در برخی از منابع با عنوان ثبات وضعیت[۲۸] از آن یادشده است) ثباتی است که بعد از انجام عملیات محاسباتی به‌وسیله ریزپردازنده، تغییر می‌کند تا امکان تعیین نوع نتیجه و تصمیم‌گیری جهت تعیین شرایط انتقال کنترل به دیگر قسمت‌های برنامه را فراهم سازد (در فصل هفتم این کتاب به بررسی ثبات پرچم می‌پردازیم).

معمولاً به همان روشی که به ثبات AX و یا سایر ثبات‌های عمومی دسترسی داریم، نمی‌توانیم به این ثبات‌ها دسترسی داشته باشیم، اما با کمی خِبرگی در برنامه‌نویسی اسمبلی می‌توان مقدار ثبات‌های سیستم را نیز تغییر دهیم.

۲-٤. تمرین فصل دوم

۱. تعداد ثبات‌های ۸ بیتی و ۱٦ بیتی را در پردازنده 8086 بیان کنید.
۲. صف دستورالعمل در پردازنده 8086 چند بایتی است.
۳. اگر DS = 5430h و SI =0021h باشد، پردازنده آدرس فیزیکی را چطور محاسبه خواهد کرد.

---

[۲٦] IP (Instruction Pointer) register
[۲۷] Flag register
[۲۸] Status register

٤. اگر بخواهیم به آدرس فیزیکی 3F85Ah (در مبنای شانزده) دسترسی داشته باشیم، آنگاه مقدار DS و SI را مشخص نمایید.

فصل ۳. دسترسی به حافظه

همانطور که در فصل قبل مشاهده کردید برای دسترسی به حافظه می‌توانیم از چهار ثبات BX، SI، DI و BP استفاده کنیم. با ترکیب این ثبات‌ها در داخل یک جفت '[ ]' می‌توان موقعیت‌های متفاوت حافظه را آدرس‌دهی کرد. روش‌های آدرس‌دهی با این ترکیبات در جدول ۳-۱ نمایش داده‌شده‌اند.

جدول ۳-۱) ترکیبات مختلف آدرس‌دهی حافظه

[BX + SI] [BX + DI] [BP + SI] [BP + DI]	[BX + SI + d16] [BX + DI + d16] [BP + SI + d16] [BP + DI + d16]	[BX + SI + d8] [BX + DI + d8] [BP + SI + d8] [BP + DI + d8]
[SI] [DI] (تنها افست متغییر) d16 [BX]	[SI + d16] [DI + d16] [BP + d16] [BX + d16]	[SI + d8] [DI + d8] [BP + d8] [BX + d8]

- d8 مشخص‌کننده یک تغییر مکان بلافصل[۱] ۸ بیتی علامت‌دار است.
  - به‌عنوان مثال 22, 55h, -1, ...

- d16 مشخص‌کننده یک تغییر مکان بلافصل ۱۶ بیتی علامت‌دار است.

[۱] immediate displacement

- به‌عنوان مثال 5517h ,250, ...

تغییر مکان‌های ۸ و ۱۶ بیتی که در بالا به آن اشاره شد می‌تواند یک ارزش بلافصل و یا افست[۲] متغیر و یا هردوی آن‌ها باشد. اگر در این حالت چندین مقدار در ترکیب آدرس قرار گیرد، مقادیر عددی جابه‌جایی را می‌توان در داخل و یا خارج زوج کاراکتر '[ ]' قرارداد. اسمبلر برای هر دو حالت، یک کد ماشین را ایجاد می‌کند. همچنین این مقادیر می‌توانند علامت‌دار (مثبت یا منفی) باشند.

۳-۱. روش ساختن آدرس مؤثر

همان‌طور که قبلاً نیز اشاره شد، ثبات سگمنت DS به‌منظور تولید آدرس مؤثر در تمام مدل‌های آدرس‌دهی (به‌جز آن‌هایی که ثبات BP دارند) مورداستفاده قرار می‌گیرد و برای روش‌هایی که از ثبات BP استفاده می‌کنند، از ثبات سگمنت SS استفاده می‌شود. برای مثال اگر SI=70 و BX=30 و DS=100 باشد و روش آدرس‌دهی 25+[BX + SI] باشد، پردازنده آدرس فیزیکی را به‌صورت زیر محاسبه می‌کند:

DS * 10h + BX + SI + d8 = 100 * 16 + 30 + 70 + 25 = 1725

یک راه ساده برای به خاطر سپردن ترکیبات ممکن در آدرس‌دهی مؤثر، استفاده از نمودار زیر است:

$$\left| \begin{array}{|c|c|} BX & SI \\ BP & DI \end{array} + displacment \right|$$

در نمودار بالا می‌توان با گرفتن یک مقدار از هر ستون و یا در نظر نگرفتن هیچ مقداری از آن ستون، تمام ترکیب‌های معتبر را بسازیم. به‌طوری‌که BX و BP و همین‌طور

[۲] offset

SI و DI هرگز باهم نوشـــته نشـــوند. برای مثال روش‌های آدرس‌دهی معتبر زیر را در نظر می‌گیریم: [BX + 5] [BX + SI] [DI + BX – 4]

تعریف سگمنت: مقادیر موجود در ثبات‌های سگمنت (ES, SS, DS, CS) را سگمنت (یا بخش) گویند. همچنین مقادیر موجود در ثبات‌های عمومی (BP, DI, SI, BX) افســـت (جابه‌جایی)، نامیده می‌شوند.

زمـانی کـه DS مقـدار 1234h و SI مقدار 7890h را داشـــته باشـــند می‌توان آن‌ها را به‌صورت 1234:7890 نوشت. به‌عبارت‌دیگر به‌جای محاسبه آدرس فیزیکی، یا آدرس موثر 1234h * 10h + 7890h = 19BD0h از نماد ':' استفاده می‌شود.

نکته: اگر یک صفر را به سمت راست یک عدد ده‌دهی اضافه کنیم در حقیقت آن را در ۱۰ ضـرب کرده‌ایم، با توجه به اینکه 10h = 16 زمانی که صـفر را به سمت راست یک عدد مبنای ۱۶ اضافه می‌کنیم مانند این است که آن را در ۱۶ ضرب کردیم.

7h = 7
70h = 112

۳-۲. تعریف نوع داده بایت و کلمه

به‌منظور مشخص کردن نوع داده در کامپایلر از پیشوندهای زیر استفاده می‌کنیم:

```
Byte ptr        ; جهت معرفی نوع داده بایت
Word ptr        ; جهت معرفی نوع داده کلمه (دو بایت)
```

برای مثال:

```
Byte ptr [BX]     ;byte access  دسترسی به بایت
Word ptr [BX]     ;word access  دسترسی به کلمه
```

اسمبلر از پیشوندهای کوتاه زیر نیز پشتیبانی می‌کند.

```
b.        ; جهت معرفی نوع داده بایت
```

```
w.      ; جهت معرفی نوع داده کلمه
```

گاهی اوقات اسمبلر نوع داده را به‌صورت خودکار محاسبه می‌کند.

۳-۳. دستور MOV

- دومین عملوند (منبع) را در اولین عملوند (مقصد) کپی می‌کند.
- عملگر منبع می‌تواند یک مقدار عددی، یک ثبات عمومی و یا مکانی از حافظه باشد.
- عملوند مقصد می‌تواند یک ثبات عمومی یا مکانی از حافظه باشد.
- هر دو عملوند باید اندازه یکسانی داشته باشند که می‌تواند بایت یا کلمه باشد.

انواع عملوندهایی که در دستورالعمل MOV پشتیبانی می‌شوند:

MOV **REG, Memory**
MOV **Memory, REG**
MOV **REG, REG**
MOV **Memory, immediate**
MOV **REG, immediate**

REG: AX, BX, CX, DX, AH, AL, BL, BH, CH, CL, DH, DL, DI, SI, BP, SP
Memory: [BX], [BX + SI + 7], variable, etc…
Immediate: 5, -24, 3Fh, 10001101b, etc…

پشتیبانی از ثبات‌های سگمنت به‌عنوان عملوند در دستور MOV به‌صورت زیر است:

MOV **SREG, Memory**
MOV **Memory, SREG**
MOV **REG, SREG**
MOV **SREG, REG**

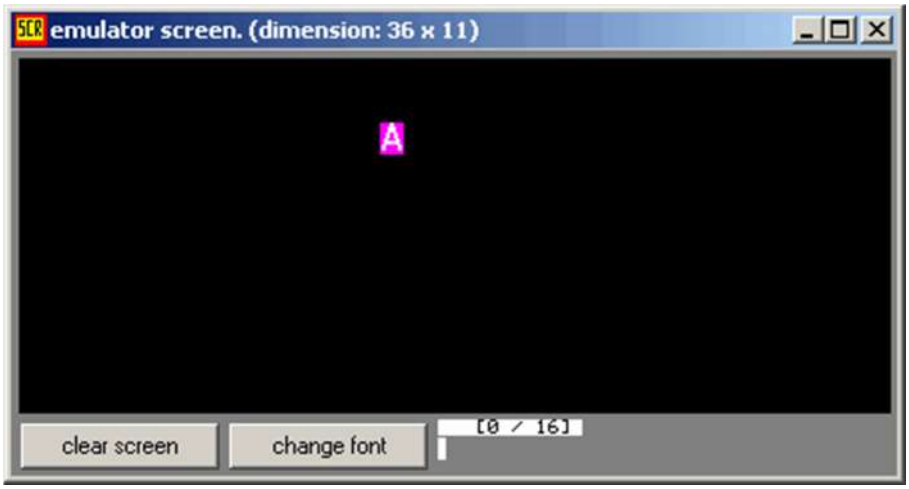
SREG: DS, ES, SS and only as second operand: CS
REG: AX, BX, CX, DX, AH, AL, BL, BH, CH, CL, DH, DL, DI, SI, BP, SP
Memory: [BX], [BX + SI + 7], variable, etc…

نکته: از دستور MOV جهت تغییر مقادیر دو ثبات CS و IP نمی‌توان استفاده کرد.

در زیر برنامه کوتاهی جهت آشنایی با نحوه استفاده از دستور MOV آورده شده است. جهت اجرای این برنامه می‌توانید متن برنامه را در داخل ویرایشگر کد شبیه‌ساز emu8086 درج و سپس دکمه [Compile and Emulate] را فشار دهید (و یا از کلید میانبر F5 جهت شروع مراحل کامپایل و شبیه‌سازی برنامه استفاده نمایید). پنجره شبیه‌ساز درحالی‌که برنامه شما در آن بارگذاری شده است بازخواهد شد. حال دکمه [Single Step] را فشار دهید (و یا کلید F8) و تغییر مقدار ثبات‌ها را مشاهده نمایید.

```
ORG 100h                ; جهت اشاره به سگمنت برنامه اجرایی comآورده شده
MOV AX, 0B800h          ; AX ← B800h  (B8000h آدرس اولین بایت حافظه ویدئویی
)
MOV DS, AX              ; DS ← AX
MOV CL, 'A'             ; CL ← 'A'    (کد اسکی 41h )
MOV CH, 1101_1111b      ; چهار بیت اولیه CH رنگ کارکتر و چهار بیت دوم رنگ پس‌زمینه
MOV BX, 15Eh            ; BX ← 15Eh (جابه‌جایی از ابتدای حافظه ویدئویی)
MOV [BX], CX            ; محتوای CX را در آدرس  B800:015E کپی می‌کند
```

همان‌طور که ممکن است حدس زده باشید علامت ';' جهت درج توضیحات برنامه‌نویس استفاده می‌شود، بنابراین تمام نوشته‌های پس‌ازاین علامت را کامپایلر در نظر نمی‌گیرد. پس از اجرایی برنامه صفحه‌ای شبیه به تصویر شکل ۳-۱ نمایش داده می‌شود.



شکل ۳-۱) خروجی اولین برنامه

در حقیقت برنامه فوق به‌طور مستقیم در حافظه ویدیویی مقادیر مورد نظر برنامه‌نویس می‌نویسد، بدین‌سان می‌توان میزان قدرت دستور MOV را تصور کرد.

۳-٤. تمرین فصل سوم

۱. با توجه به ابعاد پنجره خروجی برنامه در کامپیوتر خود (به‌عنوان مثال در شکل ۲-۱ ابعاد ذکرشده در بالای صفحه 36x11 است) رابطه عدد شانزده‌شانزدهی مقداردهی شده به BX (MOV BX, 15Eh) و محل قرارگیری کاراکتر 'A' را به دست آورید.

۲. یک جدول با ۱٦ ردیف رسم کرده و با تغییر مقدار چهار بیت بالا و پایین در CH نام رنگ مربوط به هر مقدار دودویی را یادداشت نمایید. (MOV CH, 1101_1111b)

۳. با استفاده از دو تمرین قبلی، برنامه‌ای بنویسید که نام شما را با رنگ دلخواه بر روی صفحه مانیتور نمایش دهد.

فصل ٤. متغیرها

متغیر موقعیتی از حافظه اســت. برای برنامه‌نویس بسیار ساده‌تر است که بعضی از مقادیر را در یـک متغیر با نام value ذخیره نماید تا اینکه آن را در آدرس خاصـــی از حافظه (مانند 5A73:235B) نگهداری کند، به‌خصــوص زمانی که تعداد متغیرها زیاد باشــند. این کامپایلر دو نوع از متغیرها را پشتیبانی می‌کند: byte و word

٤-١. روش اعلان یک متغیر

```
name   DB   value      ; جهت معرفی بایت DB
name   DW   value      ; جهت معرفی کلمه DW
```

نام (name)**:** می‌تواند ترکیبی از حروف (A-Z و a-z) و اعداد (0-9) باشد. به‌شرط اینکه با یک حرف آغاز شــود. امکان تعریف یک متغیر بی‌نام نیز وجود دارد. در این صــورت این متغیر نامی نداشته و تنها یک آدرس خواهد بود.

مقدار (value)**:** می‌تواند یک مقدار عددی دریکی از سـیستم‌های عددی پشتیبانی شده در emu8086 باشد (مبنای ده‌دهی، مبنای دودویی، مبنای هشت و یا مبنای ١٦) و یا می‌توانیم از نماد '؟' که نشان‌دهنده عدم ارزش‌دهی اولیه متغیر است، استفاده نماییم.

به‌عنوان مثال برنامه زیر از دستور mov برای انتقال متغیرها به ثبات‌های عمومی استفاده می‌کند.

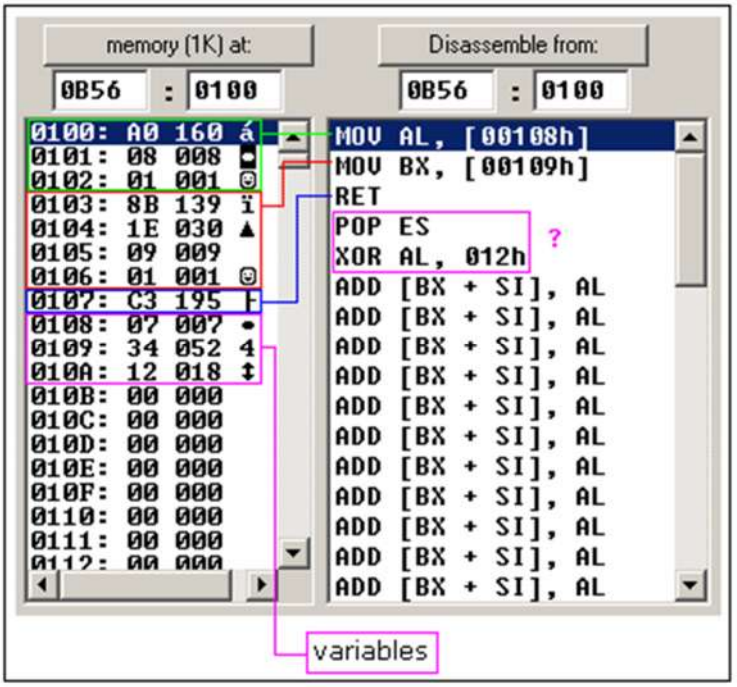
```
ORG   100h

MOV   AL,var1
MOV   BX,var2

RET             دستورالعمل توقف برنامه ;

VAR1  DB   7
Var2  DW   1234h
```

با کپی یا تایپ کد برنامه فوق در ویرایشگر منبع emu8086 و فشردن کلید F5، برنامه کامپایل و در شبیه‌ساز بارگذاری می‌شود. شما باید تصویری مانند شکل زیر را مشاهده نمایید:



شکل ٤-١) شبیه‌ساز emu8086

همان‌طور که در شکل ٤-١ مشاهده می‌کنید کدهای نمایش داده‌شده در شبیه‌ساز شباهت زیادی به کدهای برنامه دارد به‌جز اینکه در زمان ساخت کد ماشین زمانی که برنامه ارزش متغیرها را منتقل می‌کند، به‌طور خودکار نام همه متغیرها را با آفست[1] آن‌ها تعویض می‌کند. به‌طور پیش‌فرض سگمنت در ثبات DS بارگذاری می‌شود (زمانی که فایل‌های COM بارگذاری می‌شوند ارزش ثبات DS همان ارزش موجود در ثبات CS – سگمنت کد است).

نکته: در شکل ٤-١ اولین ستون در لیست حافظه آفست، دومین ستون ارزش شانزده‌شانزدهی، سومین ستون ارزش ده‌دهی، و آخرین ستون معادل کد ASCII ارزش آن خانه از حافظه است. این کامپایلر حساس به متن نیست، بنابراین "VAR1" و "var1" به یک متغیر اشاره می‌کنند. آدرس VAR1 برابر 0108h و آدرس کامل آن 0B56:0108 است. آفست var2 برابر 0109h و آدرس کامل آن 0B56:0109 است، این متغیر یک کلمه است بنابراین ۲ بایت فضا اشغال می‌کند. به‌صورت پیش‌فرض بایت کم‌ارزش‌تر در آدرس اول ذخیره می‌شود، بنابراین 34h قبل از 12h قرارگرفته است.

همان‌طور که در کد برنامه مثال قبل مشاهده می‌کنید برخی از دستورات بعد از دستور RET آمده‌اند، این بدین دلیل است که برای تبدیل‌کننده زبان ماشین به اسمبلی تفاوتی در مورد محل شروع داده وجود ندارد، تنها عملی که رخ می‌دهد پردازش ارزش حافظه و فهم این است که آن ارزش معادل دستور صحیحی در 8086 است.

برنامه قبلی را می‌توان به شکل صفحه بعد با استفاده از دستور DB بازنویسی و اجرا نمایید. همان‌طور که ممکن است حدس زده باشید پس از نوشتن این برنامه در شبیه‌ساز emu8086 و زدن کلید F5 نتایج یکسانی همانند کد اسمبلی آن حاصل می‌شود. همان‌طور

[1] offset

می‌دانید کامپایلر تنها متن برنامه را به مجموعه‌ای از بایت‌ها که به آن کد ماشین گفته می‌شود، تبدیل می‌کند سپس کد ماشین شناسایی و اجرا خواهد شد.

```
ORG 100h

DB 0A0h
DB 08h
DB 01h

DB 8Bh
DB 1Eh
DB 09h
DB 01h

DB 0C3h

DB 7

DB 34h
DB 12h
```

نکته: دستور ORG 100h یک هدایتگر کامپایلر است که به کامپایلر می‌گوید چطور باید کد منبع را مدیریت نماید. همان‌طور که مشاهده شد این هدایتگر زمانی که با متغیرها کار می‌کنیم نقش بسیار مهمی دارد. هدایتگر به کامپایلر می‌گوید که فایل اجرایی در آفست 100h (۲۵۶ بایت) بارگذاری خواهد شد؛ بنابراین هنگامی‌که کامپایلر نام متغیرها را با آفست جایگزین کرد، آدرس صحیح را برای همه متغیرها محاسبه می‌کند. هدایتگر هیچ‌گاه به کد ماشین واقعی تبدیل نخواهد شد.

نکته: چرا فایل اجرایی در آفست 100h بارگذاری می‌شود؟ سیستم‌عامل‌ها بعضی از داده‌ها در مورد برنامه را در ۲۵۶ بایت اول CS (سگمنت کد) ذخیره می‌کنند مانند پارامترهای خط فرمان و سایر موارد. به نظر می‌رسد این امر در خصوص فایل com صحیح است. فایل exe در آفست 0000h بارگذاری می‌شود؛ و معمولاً از سگمنت مخصوصی برای متغیرها استفاده می‌کند. برای درک بهتر اهمیت آدرس‌دهی درست برنامه‌ای که در فصل قبل به آن اشاره شد، با استفاده از متغیرها در زیر بازنویسی شده است:

```
ORG   100H
MOV   AX, 0B800H
MOV   DS, AX
MOV   CL, char
MOV   CH, COLOR
MOV   BX, 15EH
MOV   [BX], CL
RET
Char   DB   'A'
Color  DB   1101_1111b
```

در کد برنامه فوق ابتدا مقدار DS تغییر کرده است (DS=0B800h) در نتیجه به دلیل روش آدرس‌دهی حافظه، که در بخش قبل به آن اشاره شد، برنامه نمی‌تواند آفست متغیرها را به‌درستی بیابد؛ بنابراین برنامه به‌درستی کار نخواهد کرد.

جهت رفع این ایراد بایستی برنامه را به شکل زیر بازنویسی نماییم (تغییر محل دستوراتی که از متغیر استفاده می‌کنند):

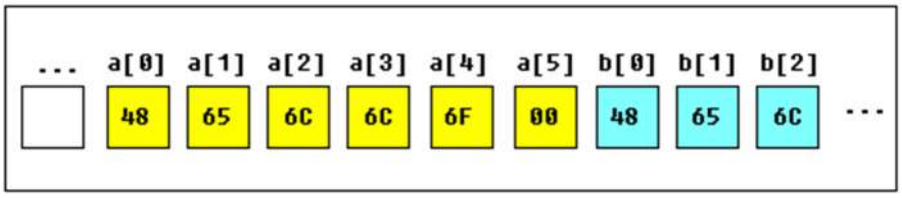
```
ORG   100H
MOV   CL, CHAR
MOV   CH, COLOR
MOV   AX, 0B800H
MOV   DS, AX
MOV   BX, 15EH
MOV   [BX], CL
RET
Char   DB   'A'
Color  DB   1101_1111b
```

٤-۲. آرایه

آرایه‌ها مانند زنجیره‌ای از متغیرها به نظر می‌رسند. یک‌رشته متنی مثالی از یک آرایه بایتی است که هر کاراکتر نماینده یک کد اسکی (0....255) است. در زیر مثال‌هایی از تعریف دو آرایه وجود دارد:

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

آرایه b دقیقاً کپی از a است، زمانی که کامپایلر به رشته‌ای برمی‌خورد که در داخل علامت نقل‌قول (quotes) قرار دارد، به‌طور خودکار آن را به مجموعه از بایت‌ها تبدیل می‌کند. تصویر زیر نشان‌دهنده قسمتی از حافظه است که آرایه a در آن تعریف‌شده است.



شکل ٤-۲) روش اعلان متغیر a و قسمتی از آرایه b در حافظه اصلی

به هر یک از بلاک‌های مربعی شکل آرایه a که در شکل بالا نمایش داده‌شده است، می‌توان دسترسی داشت. برای مثال:

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

همچنین برای دسترسی به مقدار هر عنصر از یکی از ثبات‌های BP, DI, SI, BX نیز می‌توانیم استفاده کنیم:

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

اگر نیاز دارید که یک آرایه بزرگ تعریف کنید می‌توانید از دستور DUP استفاده نمایید. در زیر شیوه صحیح به‌کارگیری دستور DUP مشاهده می‌شود:

```
number  DUP ( value(s) )
```

number: عددی که به ارزش آن، مقداری که جلوی DUP است تکرار می‌شود (که می‌توان هر مقدار ثابتی باشد).

value: عبارتی که DUP آن را تکرار خواهد کرد.

برای مثال:

```
c   DB   5   DUP(9)      =  c   DB   9,9,9,9,9
d   DB   5   DUP(1,2)    =  d   DB   1,2,1,2,1,2,1,2,1,2
```

نکته: درصورتی‌که نیاز به نگهداری مقادیری باارزش بزرگ‌تر از ۲۵۵ و یا کوچک‌تر از ۱۲۸- دارید می‌توانید به‌جای DB از DW استفاده کنید. بدیهی است که DW نمی‌تواند جهت تعریف رشته‌ها مورداستفاده قرار گیرد.

٤-۳. به دست آوردن آدرس یک متغیر

برای به دست آوردن آدرس افست یک متغیر می‌توان از دستور LEA (Load Effective Address) و یا عملگر OFFSET استفاده کرد. دستور LEA نسبت به اپراتور OFFSET بسیار قوی‌تر است زیرا اجازه می‌دهد که آدرس یک متغیر شاخص را نیز به دست آوریم.

به دست آوردن آدرس یک متغیر در بسیاری از موقعیت‌ها سودمند است. برای مثال می‌توان به انتقال پارامترها به یک رَویه[۲] اشاره کرد. مثال:

```
ORG 100h
MOV   AL, VAR1              ; جهت بررسی مقدار VAR1 را در AL قرار می دهد
LEA   BX, VAR1              ; آدرس  VAR1 را در BX قرار می دهد
MOV   BYTE PTR [BX], 44h    ; مقدار متغییر را تغییر می دهد
MOV   AL, VAR1              ; جهت بررسی مقدار VAR1 را در AL قرار می دهد
RET

VAR1  DB 22h

END
```

در مثال بعدی به‌جای دستور LEA از اپراتور OFFSET استفاده‌شده است.

[۲] procedure

```
ORG 100h
MOV   AL, VAR1              ; جهت بررسی مقدار VAR1 را در AL قرار می دهد
MOV   BX, OFFSET VAR1       ; آدرس  VAR1 را در BX قرار می دهد
MOV   BYTE PTR [BX], 44h    ; مقدار متغییر را تغییر می دهد
MOV   AL, VAR1              ; جهت بررسی مقدار VAR1 را در AL قرار می دهد
RET

VAR1   DB   22h

END
```

هر دو مثال قبل یک عمل را انجام می‌دهند. این خطوط در دو برنامه فوق:

```
LEA     BX, VAR1
MOV   BX, OFFSET VAR1
```

به‌صورت یک کد ماشین ترجمه می‌شوند: MOV BX, num که num یک مقدار 16 بیتی معادل آفست متغیر است. توجه داشته باشید که تنها ثبات‌های BX ,SI ,DI ,BP را می‌توان در داخل زوج براکت[۳] به‌عنوان اشاره‌گر به حافظه قرار داد (جهت یادآوری به بخش‌های قبل مراجعه کنید).

٤-٤. ثابت‌ها

ثابت‌ها مانند متغیرها هستند، ولی این حالت فقط تا زمانی است که برنامه کامپایل (اسمبل) شود. پس از تعریف یک ثابت مقدار آن نمی‌تواند تغییر کند. برای تعریف یک ثابت از دستور EQU به شکل زیر استفاده می‌کنیم.

name EQU < any expression >

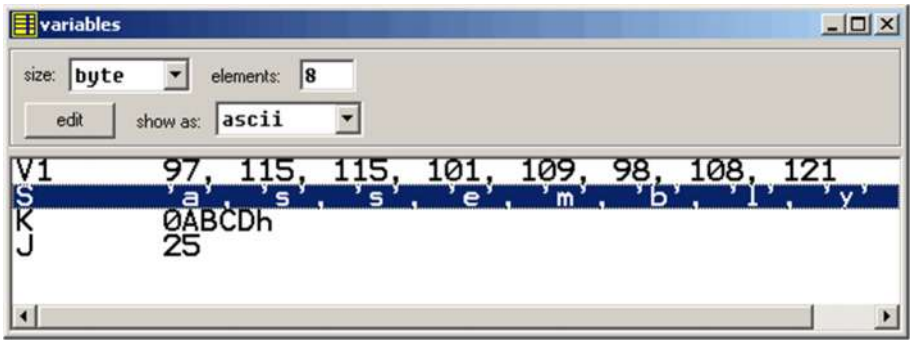
[۳] Bracket

برای مثال:

```
k EQU 5
MOV AX, 5
```

مثال قبل عملکرد یکسانی شبیه به کد زیر دارد:

٤-٥. مشاهده مقدار متغیرها در هنگام اجرای برنامه



شکل ٤-۳) پنجره Variables که از منوی View در شبیه‌ساز emu8086

برای مشاهده محتوای آرایه‌ها باید بر روی متغیر آن‌ها کلیک کرده و مقدار خصوصیت Elements را برابر اندازه هر آرایه قرار دهید. در زبان اسمبلی حساسیتی بر روی نوع داده‌ها اعمال نمی‌شود، بنابراین هر متغیر می‌تواند به‌عنوان یک آرایه معرفی شود. متغیرهای را در هر سیستم عددی می‌توان تعریف کرد، در زیر انواع متغیرها را مشاهده می‌کنید:

- HEX: شانزده‌شانزدهی (مبنای ١٦)
- BIN: دودویی (مبنای ٢)
- OCT: هشت هشتی (مبنای ٨)
- SIGNED: ده‌دهی علامت‌دار (مبنای ١٠)
- UNSIGNED: ده‌دهی بدون علامت (مبنای ١٠)

- CHAR: کد کاراکتر ASCII (در جدول اسکی ۲۵۶ نماد وجود دارد که برخی غیرقابل مشاهده هستند.)

وقتی برنامه در حال اجرا است، می‌توان مقدار هر متغیر را با دو بار کلیک کردن بر روی آن متغیر و یا انتخاب دکمه Edit ویرایش نمود. در اسمبلی امکان ورود داده در هر سیستم عددی ممکن است. برای ورود داده در سیستم عدد شانزده‌شانزدهی از پسوند 'h'، در سیستم دودویی از پسوند 'b'، در سیستم هشت هشتی از پسوند 'o' استفاده می‌کنیم و در سیستم ده‌دهی نیازی به استفاده از پسوند نیست. رشته‌ها به شکل زیر وارد می‌شوند:

```
'hello world', 0
```

توضیح این‌که برای مشخص شدن انتهای یک رشته، بایت آخر آن باید به صفر (کد اسکی NULL) ختم شود. آرایه‌ها ممکن است به این شکل تعریف شوند:

```
1, 2, 3, 4, 5
```

آرایه می‌تواند آرایه‌ای از بایت‌ها یا کلمات باشد که بستگی به انتخاب بین پیشوند BYTE یا WORD برای متغیر دارد.

عبارات به‌طور خودکار تبدیل می‌شوند، برای مثال وقتی عبارت 2 + 5 وارد می‌شود به‌طور خودکار به 7 بدل خواهد شد.

٦-٤. تمرین فصل چهارم

۱. کدام‌یک از کلمات زیر نام مجاز در اسمبلی نیست:

MUL VAr۱ VARIABLe Var.1 1.VAr FOR AVG۳

۲. چرا در برنامه‌نویسی از ثابت‌ها استفاده می‌کنیم.

۳. برنامه‌ای بنویسید محتوای دو متغییر را در ثبات AX و BX قرار داده، سپس محتوای آن دو را با یکدیگر تعویض نماید.

٤. برنامه‌ای بنویسید که محتوای دو آرایه رشته‌ای زیر را با یکدیگر عوض نماید:

```
a DB ‘assembely’,0
A DB ‘assembely’,0
```

فصل ۵. وقفه‌ها[1]

وقفه را می‌توان به‌صورت تعدادی تابع در نظر گرفت. در مقایسه با کدنویسی، این توابع کار برنامه‌نویس را بسیار آسان‌تر می‌کنند. به‌طور مثال به‌جای نوشتن کدی برای یک کاراکتر می‌توان به‌سادگی وقفه‌ای را صدا بزنیم و آن وقفه عمل چاپ کاراکتر را انجام خواهد داد.

همچنین توابع وقفه با دیسک‌خوان و سایر سخت‌افزارهای دیگر نیز کار می‌کنند. به چنین توابعی وقفه‌های نرم‌افزاری گفته می‌شود.

به وقفه‌هایی که از طریق سخت‌افزارهای متفاوت فعال می‌شوند وقفه سخت‌افزاری گفته می‌شوند. در اینجا تنها به وقفه‌های نرم‌افزاری بسنده خواهیم کرد.

برای ساختن یک وقفه نرم‌افزاری از دستور INT و شیوه ساده زیر استفاده می‌کنیم:

```
INT value
```

value یا مقدار عددی، می‌تواند بین صفر تا 255 (0 الی 0FFh) باشد.

معمولاً از اعداد در مبنای شانزده‌شانزدهی استفاده می‌کنیم. ممکن است به نظر برسد تنها 256 وقفه متفاوت وجود دارد، درحالی‌که هر وقفه می‌تواند شامل زیر روال‌های متفاوتی (حداکثر تا 256 زیر روال) باشد؛ بنابراین درمجموع تعداد روال‌های وقفه به ٦٥٥٣٦ (256 * 256 = 65536 function) خواهد رسید.

قبل از فراخوانی وقفه جهت مشخص کردن زیر تابع (زیر روال) بایستی ثبات AH را مقداردهی نماییم (اغلب از ثبات AH استفاده می‌نماییم، ولی در برخی از اوقات از سایر

[1] Interrupts

ثبات‌ها نیز ممکن است استفاده شود). عموماً از سایر ثبات‌ها جهت ارسال پارامترها و داده به زیر روال استفاده می‌شود. در مثال زیر از INT 10h (وقفه 16) و زیر روال 0Eh جهت چاپ کلمه "Hello" استفاده‌شده است. این تابع (وقفه 16 یا 10h) یک کاراکتر را چاپ کرده و مکان‌نما را به‌طرف جلو پیش برده و در صورت لزوم صفحه را حرکت می‌دهد.

```
ORG  100h  ; دستور کامپایلر جهت ساخت یک برنامه ساده یک سگمنتی از نوع .com

; زیر روال  استفاده  در این برنامه، در هنگام برگشت
; مقدار ریجیستر AH را تغییر نمی دهد، به همین دلیل
;              تنها  یکبار AH مقداردهی شده است

MOV   AH, 0Eh   ; انتخاب زیر روال

; INT 10h / 0Eh زیر روال وقفه
;  کد اسکی کارکتری که چاپ خواهد شد را
;  از ثبات AL دریافت می کند.

MOV   AL, 'H'   ; ASCII code: 72
INT   10h       ; کد اسکی موجود در AL را چاپ کن

MOV   AL, 'e'   ; ASCII code: 101
INT   10h       ; کد اسکی موجود در AL را چاپ کن

MOV   AL, 'l'   ; ASCII code: 108
INT   10h       ; کد اسکی موجود در AL را چاپ کن

MOV   AL, 'l'   ; ASCII code: 108
INT   10h       ; کد اسکی موجود در AL را چاپ کن

MOV   AL, 'o'   ; ASCII code: 111
INT   10h       ; کد اسکی موجود در AL را چاپ کن

MOV   AL, '!'   ; ASCII code: 33
INT   10h       ; کد اسکی موجود در AL را چاپ کن

RET             ;برگشت به سیستم عامل
```

جهت اجرای کد این مثال، به‌سادگی آن را در ویرایشگر کد شبیه‌ساز emu8086 کپی کرده و یا تایپ نمایید، سپس کلید [Compile and Emulate] را فشار دهید.

در صورت نیاز به اطلاعات بیشتر در خصوص انواع وقفه‌ها می‌توانید پیوست «معرفی وقفه‌های کاربردی و بردار وقفه» را مطالعه نمایید.

۵-۱. تمرین فصل پنجم

راهنمایی: جواب کلیه تمرین‌های این فصل را می‌توانید با استفاده از INT 10h و با کمک از پیوست «معرفی وقفه‌های کاربردی و بردار وقفه» بنویسید.

۱. برنامه‌ای بنویسید که یک پیام دلخواه را با استفاده از وقفه INT 10 بنویسید.
۲. برنامه‌ای بنویسید که صفحه خروجی را پاک نموده سپس در سطر ۱۰ و ستون ۲۵ کاراکتر # را (با استفاده از وقفه INT 10) چاپ نماید.
۳. برنامه‌ای بنویسید که نام خانوادگی شما را در خروجی با کاراکترهای رنگی چاپ نماید.
٤. برنامه‌ای بنویسید که اندازه صفحه‌نمایش را تغییر دهد.