ماشین حساب با زبان اسمبلی

توسط: حمیدرضا نادی مقدم

فهرست

قسمت Main

در این قسمت ابتدا صفحه نمایش با استفاده از تابع clear پاک می‌کنیم و سپس منوی اصلی را نمایش می‌دهیم. برای این کار آدرس داده منو که در menumsg را در ثبات dx قرار می‌دهیم و در ah مقدار ۹ قرار می‌دهیم. زیرا می‌خواهیم یک رشته را به نمایش در آوریم و وقفه 21h را اجرا می‌کنیم تا پیام به نمایش درآید.

سپس در ah مقدار ۱ را قرار می‌دهیم و وقفه 21h را اجرا می‌کنم و سیستم منتظر وارد کردن کاراکتری از سمت کاربر می‌ماند. بعد از فشار دادن کلیدی توسط کاربر سیستم کد اسکی آن را در al قرار می‌دهد.

در مرحله بعد انتخاب کاربر از روی صفحه کلید (کد اسکی) را با اعداد مورد نظر مقایسه می‌کنم تا کاربر را به عمل مورد نظر هدایت کنیم. برای این کار ثبات al را با دستور cmp با کد اسکی مورد نظر مقایسه می‌کنیم که در این دستور با یک عمل تفریق ساده با کد اسکی مورد نظر فلگ‌ها را تغییر می‌دهد ولی ثبات al تغییر نمی‌کند. و اگر مساوی باشد فلگ کری یک می‌شود و آن وقت با دستور je به مکان حافظه‌ای می‌خواهیم پرش می‌کنیم. در اینجا ما عدد ۱ را برای عمل جمع، عدد ۲ را برای عمل تفریق، عدد ۳ را برای ضرب، عدد ۴ را برای تقسیم، ۵ را برای توان، ۶ را برای توان عدد اویلر و صفر را برای خروج از برنامه در نظر گرفته‌ایم. اگر کاربر کلیدی غیر از این را وارد کند دوباره به تابع main را اجرا خواهد کرد.

قسمت Sum

ابتدا تابع clear را فراخوانی می‌کنیم تا صفحه نمایش پاک شود. سپس رشته‌ای برای راهنمایی به کاربر برای وارد کردن عدد اول در صفحه نمایش می‌دهیم. بعد مقدار cx را هشت قرار می‌دهیم. (یعنی کاربر می‌تواند تا هشت رقم وارد کند. توضیحات بیشتر در تابع get_operand وجود دارد). بعد از اجرای تابع get_operand که عدد مورد نظر را از کاربر گرفته است و در ثبات si و di قرار دارد را به پشته منتقل می‌کنیم تا برای خواندن عدد دوم، عدد اول از بین نرود. سپس رشته راهنمای برای وارد کردن عدد دوم را نمایش می‌دهیم. و با قرار دادن ۸ در ثبات cx با استفاده از تابع get_operand یک عدد هشت رقمی از کاربر دریافت می‌کنیم. سپس عدد اول که در پشته قرار دارد را با دستور pop در ثبات مرتبط قرار می‌دهیم. اکنون عدد اول در ثبات‌های cx (قسمت پرارزش) و bx (قسمت کم ارزش) قرار دارد و عدد دوم در ثبات‌های di (قسمت پرارزش) و si (قسمت کم ارزش) قرار دارد. بعد قسمت‌های کم ارزش دو عدد را با هم جمع می‌زنیم و قسمت‌های پرارزش را با کری قسمت کم ارزش جمع می‌زنیم. حالا با تابع print رشته‌ای

(Result) را نمایش می‌دهیم. فرخوانی تابع show نیز تمام عددهای موجود در ثبات‌های di، si، cx و bx را نمایش می‌دهد. که جمع دو عدد حداکثر ۸ رقمی که نهایتا در ثبات‌های cx و bx قرار می‌گیرد و برای همین باید di و si را با صفر مقدار دهی کنیم.

قسمت Subtract

این قسمت نیز مانند Sum است با این تفاوت که ما برای تفریق دو قسمت کم ارزش از دستور sub و برای قسمت پرارزش از دستور sbb استفاده می‌کنیم. و سپس نتیجه آن را به صورت قبل نمایش می‌دهیم و بعد از فشار دادن کلیدی از صفحه کلید توسط کاربر به منو اصلی باز می‌گردیم.

قسمت Multiply

این قسمت نیز مانند قسمت‌های قبل دو عدد را از کاربر گرفته و عدد اول را در bx (پرارزش) و bp (کم ارزش) و عدد دوم در di (پرارزش) و si (کم ارزش) قرار دارد و سپس تابع multiplication فراخوانی می‌کند. (جزئیات بیشتر در قسمت مربوطه) این تابع عدد اول را در عدد دوم ضرب می‌کند و نتیجه آن را درثبات‌های di (پرارزش‌ترین)، si، cx و bx (کم ارزش‌ترین) قرار می‌دهد. و سپس مثل قبل، نتایجی که در این ثبات‌ها قرار دارند توسط فراخوانی تابع show نمایش داده می‌شود.

قسمت Divide

در این قسمت نیز قرار دادن ۴ در ثبات cx و فراخوانی تابع get_operand عدد چهار رقمی را از کاربر می‌گیریم و در پشته قرار می‌دهیم. برای عدد دوم نیز همین کار را می‌کنیم. عدد دوم را با صفر مقایسه می‌کنیم اگر صفر بود دوباره به قسمت divide پرش می‌کنیم (برای جلوگیری از خطای تقسیم بر صفر).

عدد اول که در پشته قرار دارد را به ثبات ax منتقل می‌کنیم. و چون dx نیز در عمل تقسیم شرکت می‌کند برای احتیاط در ثبات dx را نیز صفر قرار می‌دهیم. سپس با دستور div si عدد اول (ثبات ax) را به عدد دوم (ثبات si) تقسیم می‌کنیم و جواب آن در ثبات ax (خارج قسمت) و dx (باقی مانده) قرار می‌گیرد.

سپس عدد دوم و باقی مانده را به پشته منتقل می‌کنیم. حالا با صفر کردن ثبات‌های di، si و cx و همچنین قرار دادن ax (مقدار خارج قسمت) در ثبات bx و با فراخوانی تابع print ابتدا رشته «Result» و سپس با فراخوانی تابع show مقدار خارج قسمت نمایش می‌دهیم.

سپس باقی‌مانده و عدد دوم را از پشته به ثبات‌های ax و bp به ترتیب منتقل می‌کنیم. سپس عدد باقی مانده (ax) را با صفر مقایسه می‌کنیم اگر مساوی بود (یعنی باقی مانده صفر باشد) به قسمت nofloat پرش می‌کند و اگر نه مقدار اعشار آن توسط تابع floatdiv محاسبه و توسط floatshow نمایش داده می‌شود (جزئیات این دو تابع جلوتر گفته شده است). سپس متنی به کاربر نمایش داده می‌شود و منتظر فشرد کلید از سمت کاربر می‌ماند تا به منوی اصلی بازگردد.

قسمت Euler

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!}$$

در این قسمت با قرار دادن ۲ در ثبات cx و فراخوانی تابع get_operand یک عدد دو رقمی (ثبات si) از کاربر می‌گیریم.

مرحله یک: (1+x)

عددی که از کاربر گرفته شده است را به ثبات ax منتقل می‌کنیم و یک واحد به آن اضافه می‌کنیم و در ۱۰۰۰۰ (برای بدست آوردن تا چهار رقم اعشار) ضرب می‌کنیم. و مقدار بدست آمده که در ثبات‌های dx (پرارزش) و ax (کم ارزش) هستند را به پشته منتقل می‌کنیم.

مرحله دو: (1/2! * x^2 = 15000 * x^2)

عدد که کاربر وارد کرده است (si) را دوباره به ax منتقل می‌کنیم. و در خود آن دوباره ضرب می‌کنیم. حالا در ثبات‌های dx و ax توان ۲ عدد وارد شده توسط کاربر است. چون عدد توسط کاربر حداکثر دو رقمی هست ثبات dx صفر است و جواب همان ax است که به پشته منتقل می‌کنیم. عدد به توان دو که هنوز در ax قرار دارد را با ۱۵۰۰۰ ضرب می‌کنیم. حال عدد به توان دو را که در پشته است را به ثبات bp منتقل می‌کنیم. و جواب ضرب در ۱۵۰۰۰ که در dx و ax وجود دارد را به پشته منتقل می‌کنیم. حال bp را به ax و dx را نیز خالی (صفر) می‌کنیم و در ۱۰۰۰۰ ضرب می‌کنیم. حالا جواب ضرب قبلی که در

پشته قرار دارد را از این ضرب کم می‌کنیم. قسمت‌های کم ارزش از هم کم می‌شوند و قسمت‌های پرارزش با رقم قرضی از هم کم می‌شوند. حال جواب را به پشته منتقل می‌کنیم.

مرحله سه: (1/3! * x^3 = 11666 * x^3)

در این مرحله نیز همان کارهای مرحله دوم انجام می‌دهیم با این تفاوت که عدد به توان ۲ اگه بخواهیم با خودش ضرب کنیم ممکن است عددی بیشتر شود لذا با استفاده از تابع multiplication با خودش ضرب می‌کنیم. حال عدد به توان سه را یک بار با ۱۱۶۶۶ ضرب می‌کنیمو در پشته قرار می‌دهیم و دوباره عدد به توان سه را با ۱۰۰۰۰ ضرب می‌کنیم و بعد عدد که در پشته است خارج و از ضرب قبلی کم می‌کنیم. و جواب آخر در پشته قرار می‌دهیم.

مرحله چهارم: (1/4! * x^4 = 10416 * x^4)

این مرحله نیز مانند مرحله سوم است با این تفاوت که بجای ۱۱۶۶۶ در ۱۰۴۱۶ ضرب می‌کنیم.

در آخر جواب مرحله چهارم با مرحله سوم جمع و نتیجه آن با مرحله دوم و نتیجه آن با مرحله اول جمع می‌شود. جواب مرحله‌ها در پشته قرار دارد (بالای پشته جواب مرحله سوم، بعد مرحله دوم و در پایین پشته جواب مرحله اول قرار دارد). حالا برای نمایش ثابت di را صفر می‌کنیم. و بعد پرارزش‌ترین قسمت را به ثبات si و بعد cx و در آخر کم‌ارزش ترین را به bx منتقل می‌کنیم تا به توانیم با فراخوانی تابع show عدد حاصل را نمایش بدهیم.

مرحله	عملیاتی که انجام شده	فرمول سری تیلور
مرحله اول	$1 + x * 10000$	1 + x
مرحله دوم	$15000 * x^2 - x^2 * 10000$	$\frac{x^2}{2!}$
مرحله سوم	$11666 * x^3 - x^3 * 10000$	$\frac{x^3}{3!}$
مرحله چهارم	$10416 * x^4 - x^4 * 10000$	$\frac{x^4}{4!}$

جدول ۱ نحوه محاسبه توان اویلر

ضریب اول x تقسیم یک بر فاکتوریل همان مرحله با چهار رقم اعشار و عدد که پشت آن برای حفظ اعشار آن (چون صفر ها بعد از اعشار مهم هستند) و ضریب دو هم برای کم کردن همان یک است.

مثال برای هنگامی که عدد ۵ توسط کاربر وارد شده است.

مرحله	عملیاتی که انجام شده
مرحله اول	$1 + 5 * 10000 = 60000$
مرحله دوم	$15000 * 5^2 - 5^2 * 10000$ $= 375000 - 250000 = 125000$
مرحله سوم	$11666 * 5^3 - 5^3 * 10000$ $= 1458250 - 1250000 = 208250$
مرحله چهارم	$10416 * 5^4 - 5^4 * 10000$ $= 6510000 - 6250000 = 260000$
نتیجه	$= 60000 + 125000 + 208250 + 260000 = 653250$

جدول ۲ مراحل محاسبه توان اویلر برای ۵

چهار رقم آخر اعشار هستند و رقم های بعدی آن مقدار صحیح این عدد می‌باشد. که متاسفانه به دلیل کمبود وقت نتوانستم مقدار اعشار آن را نمایش بدهم.

این جواب معمولا با جواب اصلی تفاوت زیادی دارد زیرا اولا خود سری تیلور e^x معمولا همه قسمت ها با هم جمع می‌شوند و این یعنی که هر چه مقدار مراحل یا همان حلقه بیشتر باشد به عدد واقعی نزدیکتر است و دلیل دیگر هم مقدار اعشار هست که در برنامه من فقط چهار رقم در نظر گرفته شده است.

قسمت Exit

صفحه نمایش را با فراخوانی تابع clear پاک می‌کند سپس با رشته‌ای را به نمایش در می‌آورد و با فشار دادن کلیدی، برنامه خاتمه می‌یابد.

تابع Clear

این تابع با قرار دادن ۳ در ax و اجرای وقفه 10h صفحه نمایش را پاک می‌کند. با اجرای دستور ret به مکان که تابع فراخون شده است باز می‌گردیم.

تابع Get_operand

این تابع با گرفتن کد اسکی از صفحه کلید و کم کردن آن از 30h و تبدیل به عدد کردن آن و ضرب عدد قبلی (اولین عدد صفر است) در ۱۰ و جمع آن این عدد به عدد مورد نظر کاربر که در صفحه کلید وارد کرده است می‌رسیم. اگر کاربر کلید enter را وارد کند و یا از تعداد اعداد وارد شده به اندازه تعداد در ثبات cx شود دیگر عددی خوانده نمی‌شود و عدد مورد نظر در دو ثبات di (پرارزش) و si (کم ارزش) قرار می‌گیرد. و نهایتا با اجرای ret به مکان فراخوان شده باز می‌گردیم.

ابتدا ثبات‌های si و di را صفر (مقدار اولیه) می‌گذاریم. و در ثبات bx نیز ۱۰ را می‌گذاریم. مقدار یک را ah برای خواندن یک کلید قرارمی‌دهیم و وقفه 21h را اجرا می‌کنیم. کلید وارد شده را با ۱۳ (یعنی کلید enter) چک می‌کنیم اگه مساوی بود به آدرس break می‌رویم در غیر اینصورت ah را صفر می‌کنیم ومقدار داخل al را از 30h کم می‌کنیم که مقدار خود عدد داخل al قرار بگیرد. حالا عدد را داخل پشته قرار می‌دهیم. عدد کم ارزش قبلی (si) را در ۱۰ ضرب می‌کنیم و قسمت کم ارزش آن را به si منتقل می‌کنیم و قسمت پرارزش آن را به پشته منتقل می‌کنیم. حالا قسمت پر ارزش عدد قبل (di) را با ۱۰ ضرب می‌کنیم و بعد با قسمت پرارزش قسمت قبل (si) جمع می‌کنیم. حالا عدد که توسط کاربر وارد شده که در پشته قرار دارد را با قسمت با ثبات si جمع می‌زنیم. و بعد di با صفر و کری جمع قبلی جمع می‌زنیم. سپس از مقدار cx کم می‌کنیم. و آن را با صفر مقایسه می‌کنیم اگر cx صفر شده باشد به آدرس break می‌رود در غیر اینصورت به دستور بعدی که تکرار خواندن عدد بعدی است می‌رود.

ثبات cx را می‌توان قبل از فراخوانی get_operand مقداردهی کرد تا رقم مورد نظر را از صفحه کلید بخواند.

در انتها دوثبات si (کم ارزش) و di (پرارزش) حاوی عددی که کاربر وارد کرده است هستند.

تابع Multiplication

bx-bp (msb-lsb) * di-si (msb-lsb) = di-si-cx-bx (msb to lsb)

این تابع دو ثبات bp (کم ارزش) و bx (پرارزش) را در دو ثبات si (کم ارزش) و di (پرارزش) را در هم ضرب می‌کند و جواب را در ثبات‌های di (پرارزش‌ترین)، si، cx، bx (کم ارزش‌ترین) قرار می‌دهد.

مرحله یک: ابتدا کم ارزش‌ترین ثبات‌ها یعنی bp و si را در هم ضرب می‌کنیم. سپس قسمت کم ارزش (ax) را به پشته منتقل (یک از جواب‌های اصلی) می‌کنیم و پرارزش (dx) را به cx منتقل می‌کنیم.

مرحله دوم: در مرحله bx را در si ضرب می‌کنیم و جواب قسمت کم ارزش (ax) را با پرارزش مرحله قبل که در cx قرار دارد جمع می‌زنیم. و سپس مقدار صفر را si می‌گذاریم اگر دستور jnc loo باعث پرش از روی دستور بعدی می‌شود اگر در دستور جمع قبلی کری وجود نداشته باشد (صفر باشد). در غیر اینصورت مقدار یک در si قرار می‌گیرد. با این کار مقدار کری را به پشته انتقال می‌دهیم. ابتدا مقدار cx را به پشته منتقل و سپس dx (قسمت پرارزش ضرب) را cx منتقل می‌کنیم.

مرحله سوم: در اینجا bp را با di ضرب می‌کنیم. مقدار کم ارزش (ax) را با si (جمع پرارزش مرحله دو و کم ارزش مرحله یک) که از پشته منتقل شده را جمع می‌کنیم. حال مقدار پرارزش مرحله دو (cx) و این مرحله (dx) را با کری جمع قبلی باهم جمع می‌کنیم. بعد کری مرحله قبل را پشته بیرون آورده و به ax (قسمت کم ارزش ضرب) اضافه می‌کنیم. سپس مقدار si را به پشته اضافه می‌کنیم.

مرحله چهارم: ابتدا bx را با di ضرب می‌کنیم و قسمت ax آن با cx مرحله قبل جمع می‌کنیم و به پشته منتقل می‌کنیم. در آخر dx را با کری جمع می‌کنیم و به پشته منتقل می‌کنیم.

حال در پشته پرارزش ترین در بالا و کم ارزش ترین قسمت ضرب در پایین قرار دارد.

و خروجی در ثبات‌های di، si، cx، bx قرار دارد.

bp	**bx**			
si	di	*		
ax1	dx1			bp*si
	ax2	dx2		bx*si
	ax3	dx3		bp*di
		ax4	dx4	bx*di
ax1 **(bx)**	dx1+ax2+ax3 (cx)	dx2+dx3+ax4+c (si)	dx4+c (di)	جواب نهایی
کم ارزش ترین			پرارزش‌ترین	

جدول ۳ نحوه محاسبه ضرب

تابع Print

ابتدا برای از دست ندادن مقدار دو ثبات ax و dx که در این تابع تغییر می‌کنند آنها را در پشته ذخیره می‌کنیم سپس مقدار ۹ را در ah و آفست آدرس داده resultmsg را در dx ذخیره می‌کنیم و وقفه 21h رشته را نمایش می‌دهیم. در آخر مقدار دو ثبات ax و dx سرجای آن برمی‌گردانیم.

تابع Show

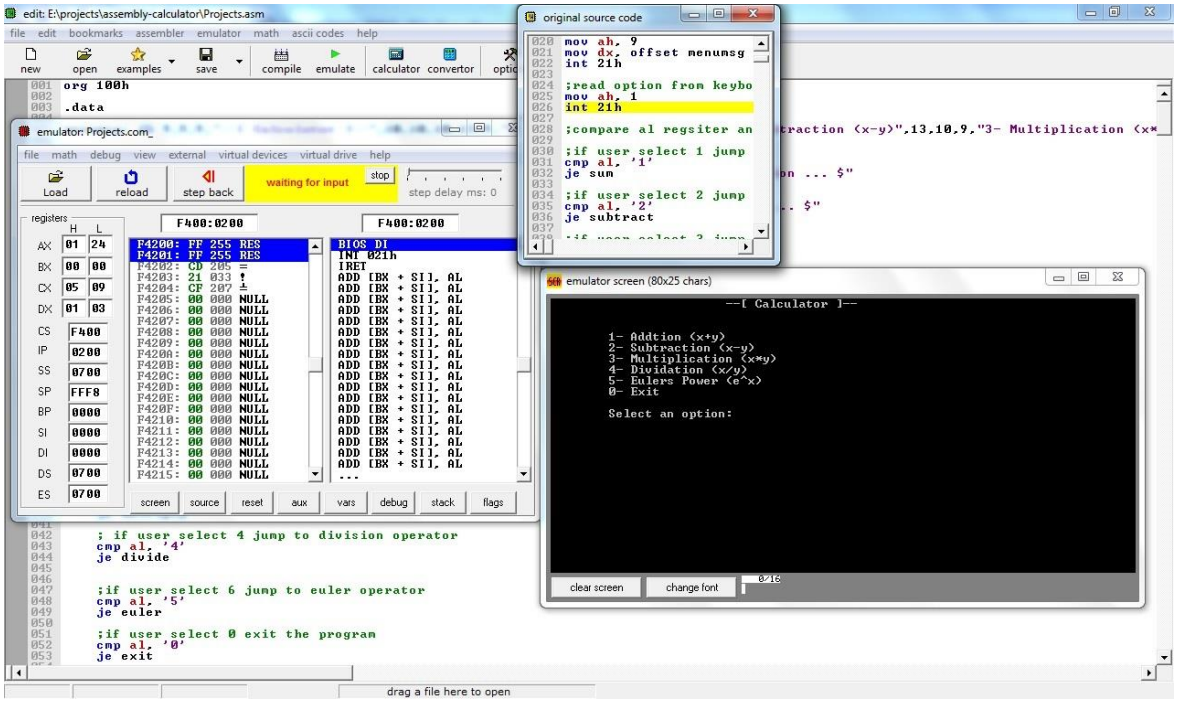
در این تابع ابتدا با تقسیم‌های متوالی بر ده مقدار ثبات bx، cx، si و di را در پشته قرا می‌دهد در پشته مقدار هر عدد به تنهای وجود دارد و پرارزش ترین مقدار در بالای پشته قرار دارد لذا برای نمایش آن کافی است عدد را از پشته خارج نماییم و مقدار 30h به آن اضافه کنیم و وقفه 21h را اجرا کنیم واین کار تا هنگامی که عدد کوچکتر از ۱۰ است تکرار کنیم. چون عدد بزگتر از ده در پشته قرار نمی‌گیرد. و در انتها با اجرای ret به جای که تابع فراخوانی شده بازگردیم.

تابع Floatdiv

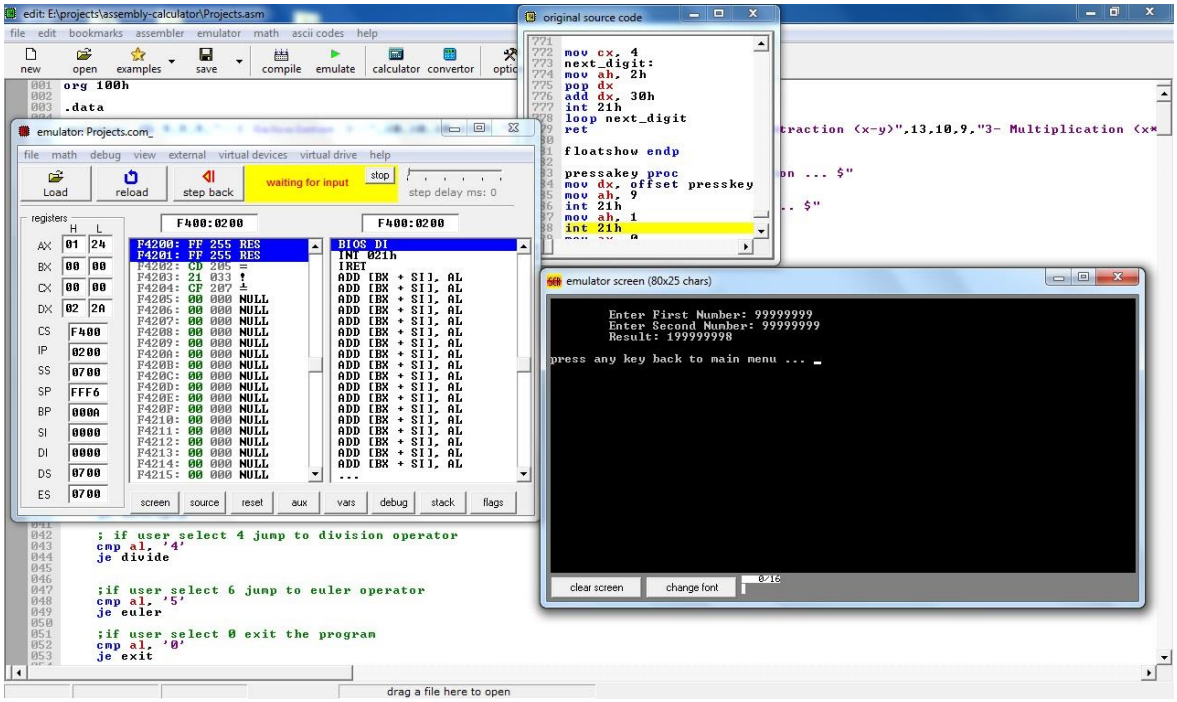
این تابع ثبات ax را به bp تقسیم می‌کند. اگر دو عدد بر هم بخش پذیر باشند (یعنی باقی مانده مساوی صفر باشد) اجرای تابع تمام می‌شود ولی در غیر اینصورت با قرار داد ۴ در cx (یعنی چهار اعدد اعشار) با ضرب در ۱۰ کردن مقدار باقی مانده و تقسیم عدد بدست آمده بر عدد دوم (یعنی bp) مقدار خارج قسمت (ax) را در پشته قرار می‌دهیم این شرط را تا صفر شدن cx و یا صفر شدن باقی مانده انجام می‌دهیم.

بعد از اتمام محاسبه اعشار بسته به اینکه ۴، ۳، ۲ و یا یک عدد در پشته باشد مقدار آن آنها در ثبات‌های منتاظر (di با کم‌ترین تا bx پرارزش‌ترین) قرار می‌دهیم. سپس مثل تابع get_oprand با این تفاوت که اولین عدد فرض شده ما یک است ۱۰ به یک اضافه می‌کنیم و سپس با bx جمع میزنیم، مجموع را با ۱۰ ضرب و با cx جمع می‌زنیم این کار را برای si و di نیز انجام می‌دهیم. در انتها جواب نهایی در ثبات bx قرار می‌گیرد و تابع با اجرای دستور ret به محل فراخوان شده بازمی‌گردد.

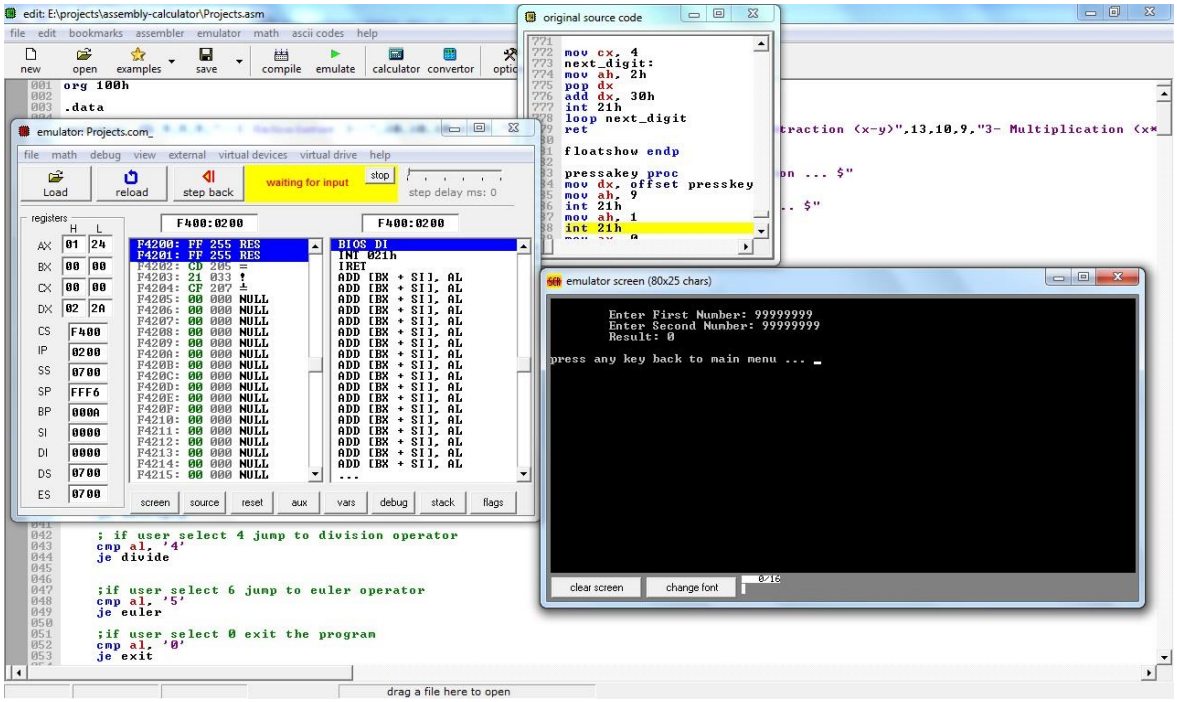
نمونه از اجرای برنامه



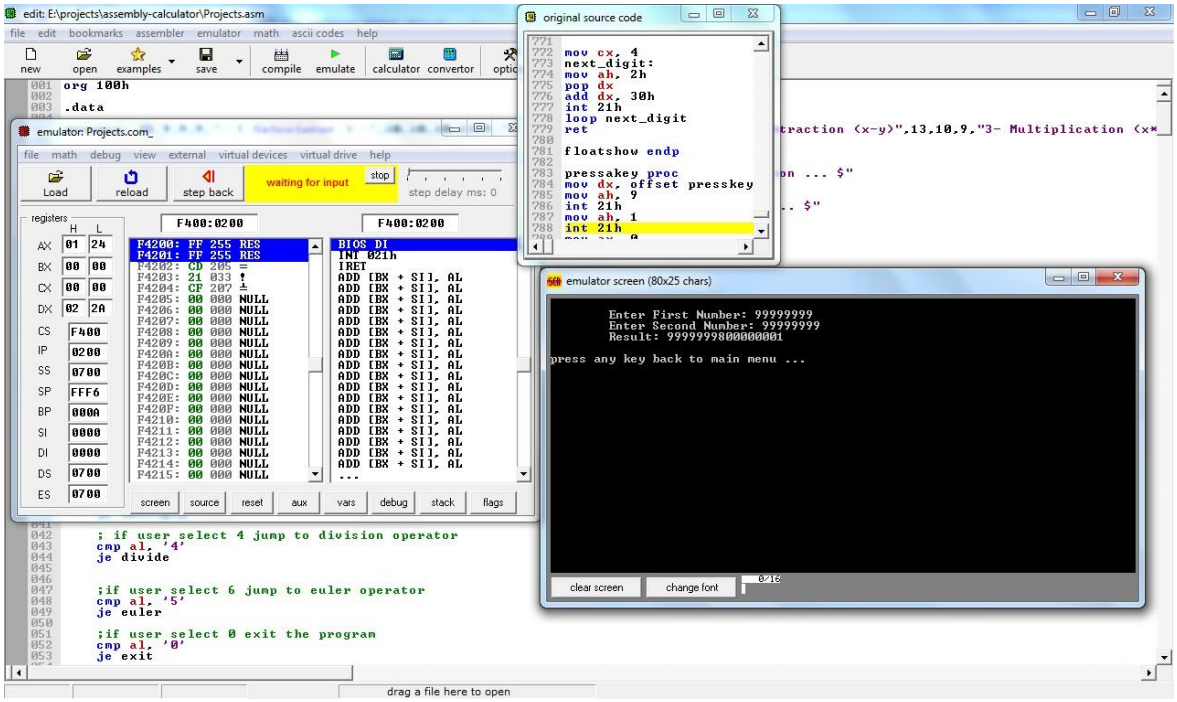
شکل ۱ منوی اصلی



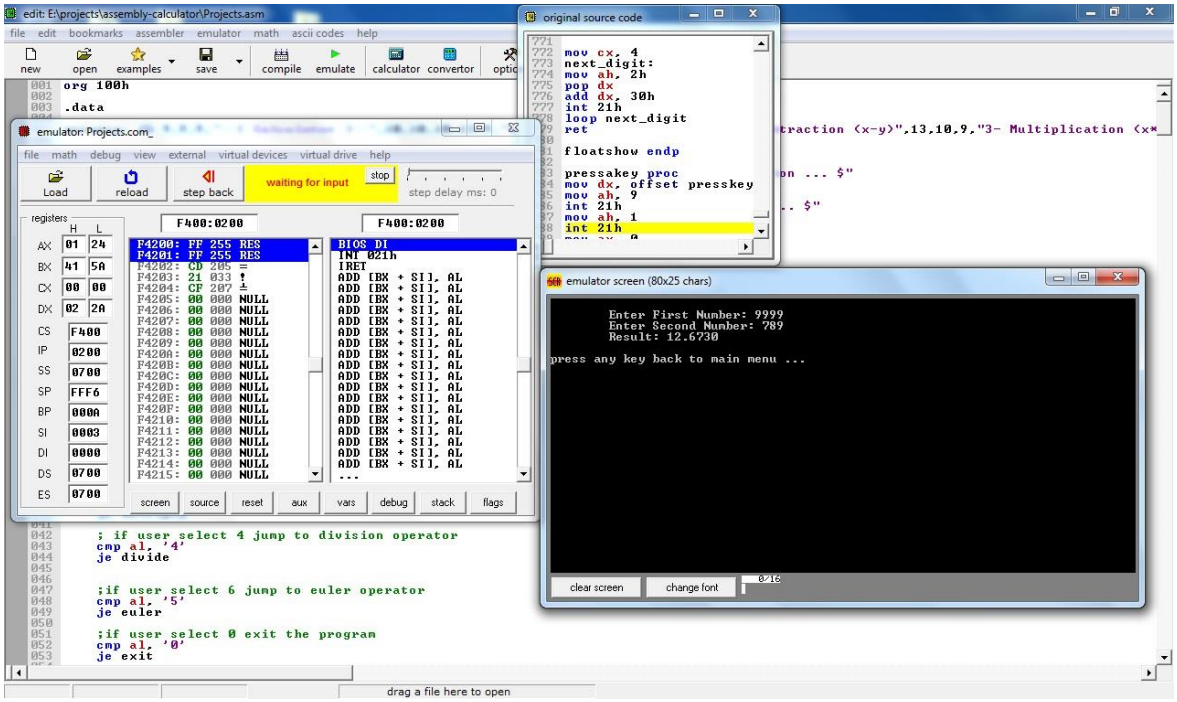
شکل ۲ نمونه جمع دو عدد



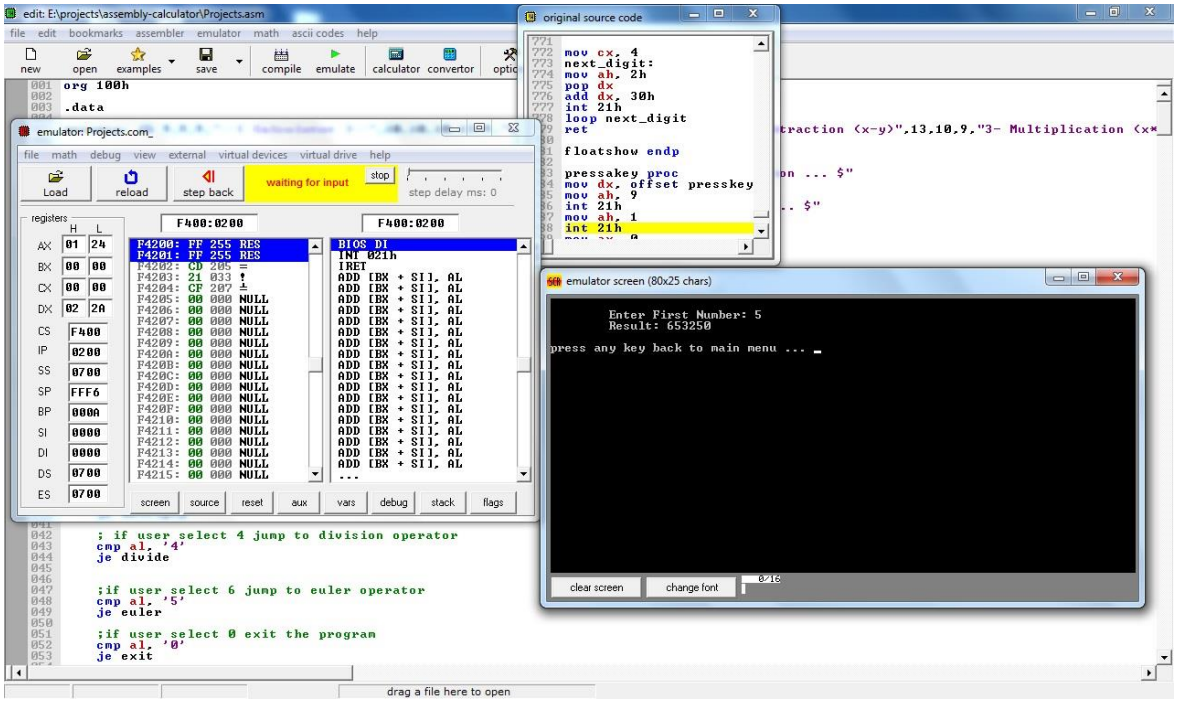
شکل ۳ نمونه تفریق دو عدد



شکل ۴ نمونه ضرب دو عدد



شکل ۵ نمونه تقسیم دو عدد



شکل ۶ نمونه عدد اویلر