به نام خدا
پروژه assembly معماری کامپیوتر

یاسمین کریما | 402243090

انجام این پروژه را به بخش‌های مختلفی تقسیم کرده و سپس همه را سرهم میکنم. بخش‌ها به ترتیب زیر هستند:

- نحوه پیاده‌سازی مسئله، رجیستر ها و چگونگی نگه‌داری مقادیر
- دریافت عدد صحیح و حلقه کلی برای دریافت دستورها
- پیاده‌سازی ۶ نوع دستور
  - دستور ۱
  - دستور ۲
  - دستور ۳
  - دستور ۴
  - دستور ۵
  - دستور ۶
- توضیح توابع استفاده شده

نحوه پیاده‌سازی مسئله، رجیستر ها و چگونگی نگه‌داری مقادیر

در مرحله اول، برای نگه‌داری اینکه چه کسی چقدر به دیگری پول داده است، ساختار جدول مناسب به نظر میرسد. توضیح بیشتر خواهم داد.

همچنین برای سرچ آسان در میان نام ها، بهتر است یک آرایه تحت عنوان names داشته باشیم که در آن هر نام متناظر با یک اندیس است؛ در جدول به جای استفاده از نام(رشته) از این اندیس ها استفاده میکنیم.

علاوه براینها، خوب است سرمایه کلی هر فرد را را همیشه به روز داشته باشیم. پس یک آرایه برای سرمایه ها داریم که با هر دستور نوع اول باید مقدار لازم در آن به روزرسانی شود.

جدولی که گفتم مشابه زیر است:

1 s3 s1 5.00 / 1 s1 s3 3.00 / 1 s4 s3 6.01

	۱	۲	۳	۴
۱	-	0	4.00	0
۲	0	-	0	0
۳	5.00	0	-	0
۴	0	0	6.01	-

با دستور نوع اول p1 p3 4.00، خانه [1 ,3] مقدار 4.00 را میگیرد.با دستور نوع اول p3 p1 5.00، خانه [3 ,1] مقدار 5.00 را میگیرد. برای محاسبه اینکه ۱ چقدر به ۳ بدهکار است:

[3, 1] – [1, 3]

برای محاسبه اینکه ۱ چقدر از ۳ طلبکار است:

[1, 3] - [3, 1]

و مقدار منفی نشان دهنده طلبکار نبودن است.

همچنین برای جلوگیری از خطاهای اعداد floating point، من اعداد را به شکل int ذخیره میکنم؛ ورودی ضرب در ۱۰۰ می‌شود و برای نمایش خروجی، تقسیم بر ۱۰۰.

همچنین بخشی را نیز برای اجزای دستور در نظر میگیریم(tokenized input)، یعنی نام و مقادیری که در دستور آمده. لیبل temp_tokens برای این منظور است.

متغیر instr از جنس رشته را برای نگه داری دستورات تعریف میکنیم. بیشترین طول دستور مربوط به دستور نوع اول است که ۶۴ بایت برای آن کافی است.

تعداد آدم‌ها را حداکثر ۱۰۰ در نظر میگیرم و بخش data کامل میشود.

دریافت عدد صحیح و حلقه کلی برای دریافت دستورها

ابتدا q که تعداد دستورات را نشان میدهد را ورودی میگیریم. ورودی از جنس int است که در اسمبلی syscall = 5 میباشد. سپس این مقدار q داخل رجیستر s3 ریخته می‌شود. لازم است روی آن حلقه ای اجرا کنیم که در آن مقدار s3 کم می‌شود و وقتی s3 = 0 کار خواندن دستورات تمام می‌شود و برنامه را تمام میکنیم.

در این حلقه باید دستورات که به صورت رشته هستند را ورودی بگیریم syscall = 8 مخصوص ورودی رشته است، البته باید a0 و a1 مقداردهی شوند، a0 با آدرسی که رشته در آن نوشته می‌شود و a1 با حداکثر بایت که باید خوانده شود.

سپس اجزا ورودی را جدا میکنیم و با توجه به کاراکتر ابتدای دستور، نوع آن مشخص میشود.

پیاده‌سازی دستور نوع ۱: s1 s2 x

فرد s1 به فرد s2، مقدار x دلار پول داد.

ابتدا با تابع is_name_registered بررسی میکنیم آیا اسم مورد نظر تا به حال در names ذخیره شده یا خیر، در صورت عدم وجود، آن‌ها را اضافه میکنیم. پس از این مراحل، طی توابعی که فراخوانی کرده ایم، مقادیر اندیس افراد را داریم.

حالا باید مقدار x که به فرمت مشابه 45.23 است را به integer تبدیل کنیم. از تابع extract_money استفاده میکنم.

پس از آن در table تغییر لازمه را ایجاد میکنیم، x دلار از سرمایه s1 کم و x دلار به سرمایه s2 اضافه میکنم.

تغییرات لازمه تمام شده و به حلقه اصلی برمیگردیم.

تابع add_name:

قرار است یک نام به names اضافه شود.با استفاده از تعداد افراد، محل خالی در آرایهnames را پیدا میکنم و باcopy_name نام جدید را در آنجا مینویسم. تعداد افراد هم باید یکی اضافه شود.

همچنین درنظر داریم که ما برای نام هر شخص به جای ۸ بایت، ۱۲ بایت در نظر گرفتیم، زیرا بعد از هر نام به اندازه یک word جای خالی گذاشتیم تا در زمان پرینت نام ها به مشکل نخوریم.

تابع copy_name:

دو پوینتر میگیرد، یکی برای نام مورد نظر و دیگری برای محل کپی کردن؛ ۸ بایت را بررسی میکند و در مقصد کپی میکند.

پیاده‌سازی دستور نوع ۲:

لازم است در آرایه سرمایه ها بیشترین مقدار را پیدا کنیم. در آرایه میگردیم، اگر مقداری اکیدا بیشتر از base داشتیم، جایگزین میکنیم و اندیس آن را نگه میداریم.

اگر مقدار بعدی مساوی بیشترین مقدار(base) بود و base !=0 ، لازم است اندیس شخصی را نگه داریم که نامش از نظر لغتنامه ای کوچکتر است. این مطلب را با تابع compare_Lexicographically بررسی میکنم.

در انتها اگر اندیس نهایی همچنان -1 باشد، یعنی کسی سرمایه مثبت ندارد، وگرنه شخص ثروتمند توسط اندیسش پیدا می‌شود و نام او پرینت خواهد شد.

پیاده‌سازی دستور نوع ۳:

کاملاً مشابه درستور ۲، فقط کافی است مقادیر کوچکتر از base را با آن جایگزین کنیم.

پیاده‌سازی دستور نوع ۴:

با فراخوانی تابع is_name_registered اندیس نام ذکر شده را خواهیم داشت.

در ادامه آنچه در روش پیاده‌سازی توضیح دادم، برای یافتن تعداد افرادی که شخص به آن‌ها بدهکار است، لازم است ستون مربوط به آن شخص را بگردیم؛ زیرا وقتی کسی به او پول میداد، در ستون شخص و سطر پول دهنده آن را ذخیره میکردیم. اما از آنجاییکه ممکن است شخص بدهکاری اش را تسویه کرده باشد، باید درایه متقارن را نیز بررسی کنیم و مقدار بدهکاری را با تابع debt_value به دست آوریم. در صورت مثبت بودن شخص بدهکار است و counter را زیاد میکنیم.

سپس برای نفر بعدی در ستون شخص، آدرس را ۴۰۰ تا جلو میبریم. هر ردیف یک آرایه ۱۰۰ تایی از int های 4 بایتی است.

پیاده‌سازی دستور نوع ۵:

در این دستور میخواهیم تعداد افرادی را بیابیم که شخص از آن‌ها طلب دارد، پس باید ببینیم تا به حال به چه کانی پول داده؛ پس سطر متناظر با شخص را بررسی میکنیم و به ازای هر درایه مانند دستور ۴، چک میکنیم آیا طلبکاری تسویه شده یا خیر(با همان تابع debt_value).

پیاده‌سازی دستور نوع ۶: s1 s2

ببینیم s1 چقدر باید به s2 بدهد. در روش پیاده‌سازی توضیح دادم که باید از جدول table مقدار خاصی استخراج شود و با تابع debt_value مقدار بدهی مشخص شود. توضیح این بخش در کامنت های کد تقریباً واضح است :).

همچنین در انتها باید عدد integer را به فرمت مورد نظر تبدیل کنیم. این هم در کد توضیح دادم.

توضیح توابع استفاده شده: (در utils.s)

- **Make_zero**

برای خالی کردن مکان های موقتی مانند temp_tokens.

- **tokenize_input**

قرار است دستور ورودی به نام ها و مقدار هایش بشکند، کاراکتر به کاراکتر پیش می‌روم و با توجه به تعداد space ها تشخیص میدهم قرار است نام اول را ذخیره کنم، نام دوم را ذخیره کنم یا به قسمت مقدار پول رسیده ام. برای دستورات غیر نوع ۱، قبل از این‌ها دستور تمام شده و به لیبل done رفته ایم.

name_equal

دو پوینتر به دو نام ورودی میگیرد و کاراکتر به کاراکتر تساوی آن‌ها را بررسی میکند. اگر ۸ کاراکتر بررسی شد و تمام حروف یکسان بود، نام ها یکی هستند، وگرنه یکی نخواهند بود.

مشکل: گاهی اوقت پوینتر به ورودی از سیستم بود در انتهای ورودی \n داشتیم، حتی اگر اسم‌ها یکی بودند، به خاطر وجود \n در انتهای یکی، نام ها نامساوی در نظر گرفته میشدند، پس \n را به 0 تبدیل کردم.

is_name_registered

بررسی میکند نام ورودی تا به حال در names بوده؟ اگر بله اندیس آن را برمیگرداند و در غیر اینصورت -۱ بر میگرداند.

extract_money

در دستور نوع ۱، در انتهای دستور یک مقدار پول داشتیم که به شکل رشته ذخیره شده بود، باید آن را به صورت عدد صحیح دربیاوریم. روش آن است که به ازای هر رقم از سمت چپ، یک base را در 10 ضرب کرده و با رقم فعلی جمع کنیم و از ممیز صرف نظر کنیم.

compare_Lexicographically

اندیس دو نام را میگیرد و بررسی میکند از نظر الفبایی کدام کوچکتر است، مشخصاً باید کاراکتر به کاراکتر جلو برود و ببیند کدام کاراکتر کوچکتر از دیگری است.

ار طول یک نام از دیگری کمتر باشد، در انتهای آن 0 قرار دارد، وقتی به کاراکتر بعد از آخرین برسیم، 0 با یک کاراکتر الفبایی مقایسه می‌شود و کد اسکی 0 از تمام حروف الفبا کمتر از، پس نام با طول کمتر، کوچکتر از نظر الفبایی محسوب خواهد شد.

debt_value

ما در مواقع مختلف، مقدار یک درایه [b,a] را داریم و برای هدفمان، مقدار [a, b] را لازم داریم. این تابع اختلاف مورد نظر را که مقدار بدهی یا طلب است، برمیگرداند.

اگر بخواهیم بدانیم b چقدر از a طلب دارد:

[b, a] – [a, b]

اگر بخواهیم بدانیم b چقدر به a بدهی دارد:

[a, b] – [b, a]

تصاویر از اجرای برنامه