

<p style="text-align: center;"><u>Wykonał:</u></p> <p style="text-align: center;">Bystryk Dominik (Nr Indeksu: 254017)</p>		<p>Wydział: Informatyka Techniczna Filia JG</p> <p>Rok: 2</p> <p>Nr grupy: 1</p> <p>Rok akadem.: 2023/2024</p>
Organizacja i architektura komputerów (laboratorium)		
<p>Data zakończenia projektu: 24.04.2024 r.</p>	<p><u>Temat ćwiczenia laboratoryjnego:</u></p> <p style="text-align: center;"><i>Konwersja wybranego algorytmu na funkcję. Ramka stosu. Wywołanie funkcji rekurencyjnie.</i></p>	

Streszczenie – Ćwiczenie polegało na zamianie wybranego algorytmu z wcześniej napisanego kalkulatora dużych liczb heksadecymalnych na funkcję wywoływaną za pomocą instrukcji *call* oraz na wykorzystanie stacka do transportu danych do funkcji (łącznie ze zmianą ramki stosu).

1. Wykorzystane oprogramowanie

- Virtual Box ver. 7.014
- Ubuntu 23.10
- Assembler NASM ver 2.16.01
- Text Editor wbudowany w OS

2. Cel i zakres ćwiczenia oraz opis sposobu wykonania ćwiczenia.

Celem ćwiczenia było wybranie z poprzednio napisanego projektu jednego z algorytmów działania algebraicznego na 100 bajtowych liczbach hexadecymalnych, przerobienie go na funkcję wywoływaną za pomocą instrukcji *call* oraz przekazanie argumentów do tego działania za pomocą stosu (stacka) wraz z przesunięciem ramki stosu z pomocą instrukcji *enter* i *leave*.

3. Wstęp teoretyczny

Procedura *call (return)* oraz *enter (leave)* (w połączeniu z procedurą *call (return)*) wykorzystują stos do zapisu stanu procedury, która wywołuje, przekazania parametrów do funkcji wywoływanej i do zapisu lokalnych zmiennych, których używa wywoływana funkcja.

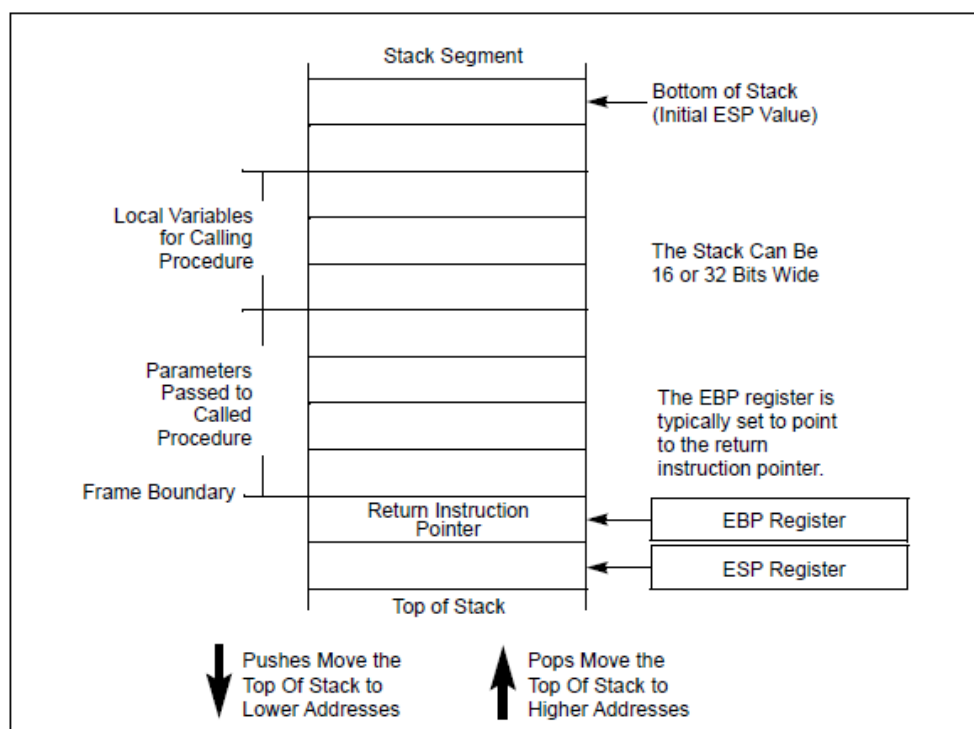
W ogólnym pojęciu stos (stack) jest ciągłą tablicą zawierającą lokalizacje w pamięci. Składa się z segmentów, które są rozpoznawane dzięki rejestrowi SS (stack segment register). Każdy segment może maksymalnie posiadać długość do 4 GB.

Do obsługi stosu służą dwie instrukcje: *push* i *pop*.

Instrukcja *push* kopiuje zawartość np. rejestru na szczyt stosu, podczas gdy instrukcja *pop* ściąga (zabiera) wartość ze szczytu stosu i umieszcza ją np. w rejestrze.

Za każdym razem kiedy umieszczamy element na szczycie stosu rejestr wskaźnikowy stosu (ESP) jest dekrementowany (idzie w kierunku niższego adresu), podczas gdy ściągnięcie ze szczytu stosu powoduje inkrementację ESP (idzie w kierunku wyższego adresu). Dlatego mówi się, że stos w języku assembly rośnie w dół a kurczy się w górę. W obrębie stosu istnieją dwa rejestry wskaźnikowe – EBP (stack-frame base pointer) oraz ESP (stack pointer).

Wskaźnik podstawy ramki stosu stanowi trwale miejsce odwołania dla wywoływanej za pomocą funkcji *call* instrukcji. Po przeniesieniu wskaźnika EBP (równoznacznego z przeniesieniem ramki stosu) nie ma możliwości przypadkowego nadpisania przekazywanych danych lub nadpisania adresu powrotu do funkcji wywołującej.



Ryc 1: Przykładowy obraz stosu

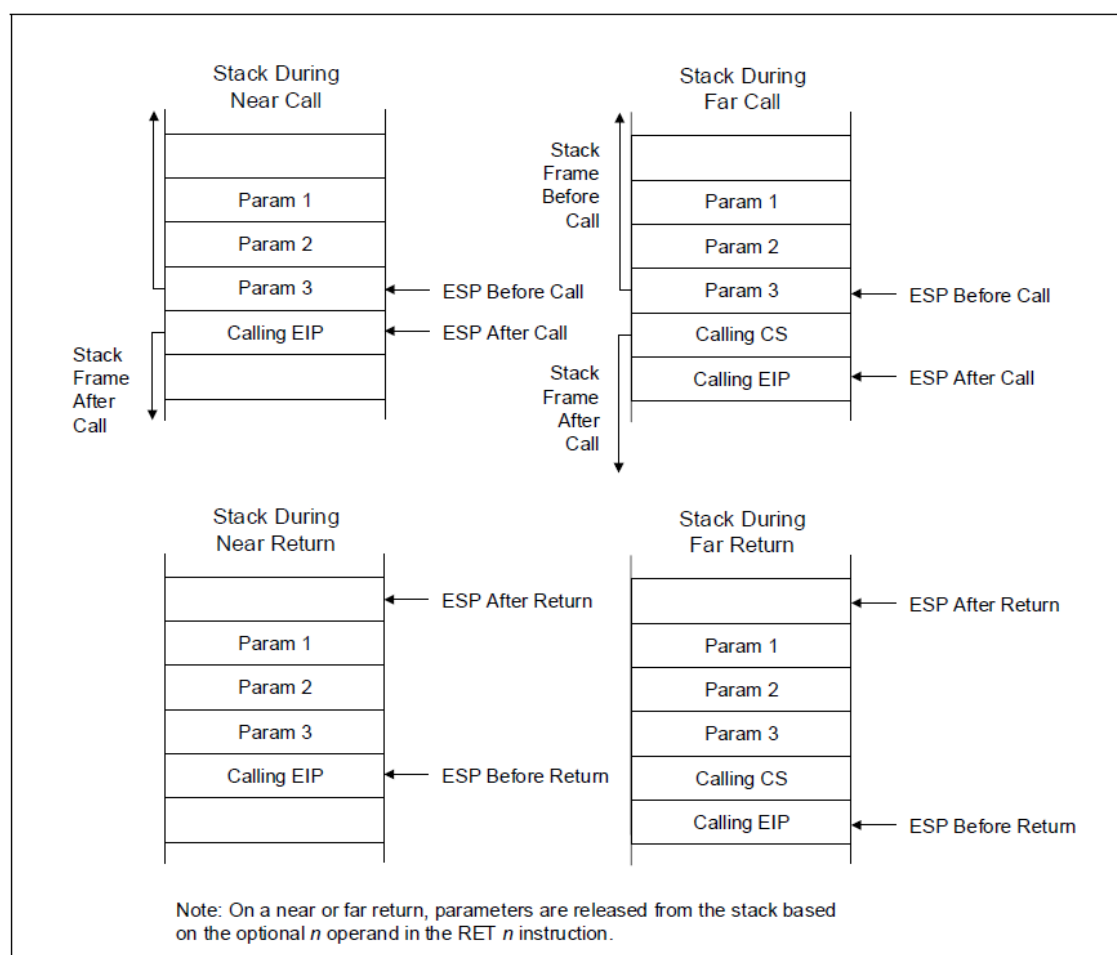
4. Działanie instrukcji *call* (*return*)

Przed przejściem do funkcji wywoływanej. Funkcja *call* odkłada na szczycie stosu adres zawarty w rejestrze EIP (*push*). Jest to adres, który informuje funkcję, po dojściu do instrukcji *ret*, w które miejsce programu ma wrócić po skończeniu działania na wywoływanej funkcji. Natknięcie się na instrukcję *ret* spowoduje zabranie tego adresu ze szczytu stosu (*pop*) i umieszczenie ponownie w rejestrze EIP; program będzie kontynuował działanie od miejsca po „*call*’u”.

Uwaga ! Procesor nie zapamiętuje adresu powrotu. Programista sam musi zadbać o to, by program miał dostęp do prawidłowego adresu powrotu.

Powrót instrukcją *ret* nie musi powodować powrotu do miejsca wywołania. Można dowolnie manipulować, do którego adresu ma nastąpić powrót; można wykonać powrót do adresu w danym segmencie kodu (*near return*) lub do adresu w innym segmencie kodu (*far return*).

Ryc 2: Wygląd stosu przy *near/far call*’u.



5. Przekazywanie parametrów funkcji

Parametry dla funkcji wywoływanej można przekazywać w następujący sposób:

a) poprzez rejestr ogólnego przeznaczenia

Ponieważ procesor nie zapisuje zawartości rejestrów ogólnego przeznaczenia (wyjątkiem są rejestry ESP i EBP) korzystając z nich można przekazać argumenty do funkcji oraz

otrzymać wyniki działania wywoływanej funkcji.

b) poprzez stos

c) poprzez listę argumentów (np. tablicę)

Odbywa się to poprzez przekazanie wskaźnika listy przez stos lub jeden z rejestrów ogólnego przeznaczenia.

6. Działanie instrukcji *enter (leave)*

Instrukcja *enter (leave)* ogólnie działa podobnie jak funkcje *call (ret)*, aczkolwiek różni się w kilku kluczowych aspektach:

- instrukcja *enter(leave)* zajmuje się obsługą ramki stosu – tworzy ją i zwalnia, rezerwuje miejsce na lokalne zmienne i argumenty.

- instrukcja *enter* pozwala na dynamiczną rezerwację bajtów na stosie dla wywoływanej funkcji (pierwszy operand; dla argumentów tworzonych przy wywoływaniu funkcji), podczas gdy drugi parametr odpowiada za poziom zagnieżdżenia procedury.

Instrukcja *leave* zdejmuje całą ramkę utworzoną przez instrukcję *enter*.

7. Wykonanie zadania.

a) Funkcja *liczenie* została zmodyfikowana:

- poprzez 4 rejestry ogólnego przeznaczenia (*eax, ebx, ecx, edx*) przekazuje na stos (*push*) wskaźniki na 1. argument tablic potrzebnych do wykonania funkcji sumowania.

- dochodzi do wywołania funkcji *liczsum* za pomocą instrukcji *call*

Przybliżone przedstawienie stosu przed użyciem instrukcji *call*

EBP
PrzepisanaLiczba_arr
Wynik_arr
Num2_arr
Num1_arr
*Tutaj wskazuje ESP

b) Funkcja *liczsum* została zmodyfikowana:

- tworzy ramkę stosu na rzecz funkcji sumowania za pomocą instrukcji *ENTER 0,0* (nie rezerwujemy dynamicznie pamięci)

c) Funkcja *done* została zmodyfikowana:

- dodano instrukcję *leave* celem zniszczenia ramki stosu przeznaczonej dla funkcji sumowania i przywrócenia wskaźnika *EBP* do postaci poprzedniej.

- dodano instrukcję *ret* by powrócić do funkcji *liczenie*, która wywoływała funkcję odpowiedzialną za przeprowadzenie dodawania.

W podobny sposób zmodyfikowano funkcję *strlen1dl*:

- dodano instrukcję *leave* celem zniszczenia ramki stosu przeznaczonej dla funkcji sumowania i przywrócenia wskaźnika *EBP* do postaci poprzedniej.

- dodano instrukcję *ret* by powrócić do funkcji *liczenie*, która wywoływała funkcję odpowiedzialną za przeprowadzenie dodawania.

8. Rekurencja.

Podjęto także próbę stworzenia rekurencyjnego wariantu wykonania funkcji dodawania. Niestety kod z dnia 24.04.24 r. działał niepoprawnie.

W trakcie przerwy majowej udało mi się dodatkowo pozmienić kod funkcji, by stworzyć funkcję dodawania w postaci wywołania rekurencyjnego.

Wprowadzone zmiany względem kodu funkcji dodawania bez rekurencji:

a) funkcja *sumator* stała się funkcją sprawdzającą czy zakończono procedurę dodawania

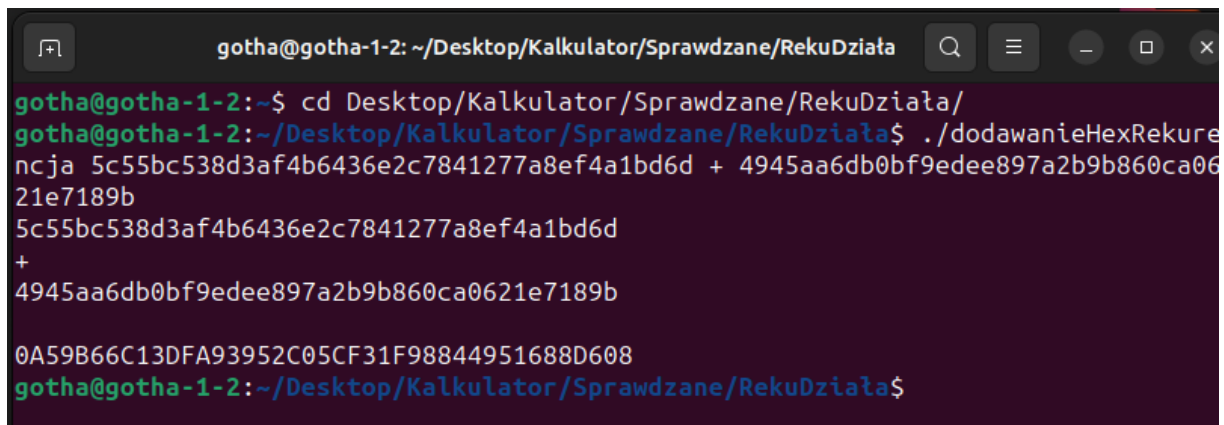
oraz wywołującą funkcję rozpoczynającą procedurę dodawania – *add_podstawowy*

b) funkcja *add_podstawowy* jest funkcją sprawdzającą czy zakończono procedurę dodawania oraz przeprowadza dodawanie

c) funkcja *add_loop* stała się funkcją odpowiedzialną za prawidłowe dostosowanie argumentów przechowywanych w rejestrach ogólnego przeznaczenia, które wcześniej zostały pobrane ze stosu; wyjątkiem jest wskaźnik do tablicy *Wynik_arr*, który, z powodu specyfiki zaimplementowanego algorytmu, w tym miejscu jest odtwarzany niezależnie czy doszło do przeniesienia, czy też nie (pobierany ze stacka wskaźnik na 1. element tablicy, a następnie dostosowywany przez dodanie zawartości rejestru *ecx* do rejestru przechowującego adres wskaźnika).

Dodatkowo funkcja ta ma za zadanie rekurencyjnie wywołać procedurę dodawania – *call add_podstawowy*.

d) dodano funkcję *add_koniec*, która ma za zadanie powrócić w odpowiednie miejsce, by zakończyć całą procedurę dodawania (wrócić do funkcji *sumator*, by przejść do zakończenia procedury).



```
gotha@gotha-1-2: ~/Desktop/Kalkulator/Sprawdzane/RekuDziała
gotha@gotha-1-2:~/Desktop/Kalkulator/Sprawdzane/RekuDziała$ ./dodawanieHexRekurencja 5c55bc538d3af4b6436e2c7841277a8ef4a1bd6d + 4945aa6db0bf9edee897a2b9b860ca0621e7189b
5c55bc538d3af4b6436e2c7841277a8ef4a1bd6d
+
4945aa6db0bf9edee897a2b9b860ca0621e7189b
0A59B66C13DFA93952C05CF31F98844951688D608
gotha@gotha-1-2:~/Desktop/Kalkulator/Sprawdzane/RekuDziała$
```

Ryc 3: Przykład wyniku dla rekurencyjnego wywołania procedury dodawania

9. Literatura

[1] notatki i materiały z wykładu oraz laboratoriów z przedmiotu „Organizacja i architektura komputerów”, źródło niepublikowane

[2] „Intel® 64 and IA-32 Architectures Software Developer’s Manual Combined Volumes: 1, 2A, 2B, 2C, 2D, 3A, 3B, 3C, 3D, and 4”