Wykonał: Bystryk Dominik	Wydział: Informatyka Techniczna Filia JG Rok: 2 Nr grupy: 1 Rok akadem.: 2023/2024
Organizacja i architektura komputerów (laboratorium)	
Data zakończenia projektu: **26.03.2024 r.**	**Temat ćwiczenia laboratoryjnego:** ***Kalkulator dużych liczb heksadecymalnych***

Streszczenie – Ćwiczenie polegało na przygotowaniu programu, który dla dwóch dużych liczb A i B w pozycyjnym zapisie szesnastkowym wykonywałby operacje dodawania, odejmowania i mnożenia. Program napisany w języku assembly na procesor x86

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 przygotowanie programu w języku assembly, który miał, dla dwóch dużych (do 100 bajtów) liczb w zapisie pozycyjnym szesnastkowym wykonywać działania algebraiczne t.j. dodawanie, odejmowanie oraz mnożenie.

   Dodatkowo program miał pobierać argumentu z wiersza poleceń i działać na architekturze x86 (32-bit).
   Kompilacja (assemblowanie) programu miało zostać zautomatyzowane za pomocą pliku Makefile.

   Ze względu na brak wcześniejszego kontaktu z językiem assembly, po wstępnym zaznajomieniu się z podstawami języka przygotowano następujący sposób wykonania ćwiczenia:
   a. Przygotować program pobierający argumentu z wiersza poleceń i wypisujący je na ekranie
   b. Do wyżej wymienionego programu przygotować możliwość wpisania tychże argumentów (tylko liczb A i B oraz znaku) do odpowiednich tablic, alokowanych w pamięci przydzielanej programowi.
   c. Dopisać podstawowe porównanie, od którego będzie zależało dalsze działanie programu (porównanie wartości argumentu „znak” do odpowiedniego znaku algebraicznego)
   d. Po prawidłowym przyrównaniu znaku przekonwertować „półbajt po

półbajcie” tabele zawierające liczby z ASCII na hex
e. Wykonanie działania arytmetycznego zgodnie z wprowadzonym wcześniej znakiem arytmetycznym i wpisanie wyniku do tabeli „wynik”
f. Przekonwertowanie elementów tablicy „wynik” z systemu Hex na ASCII i wypisanie ich w terminalu.
g. Wprowadzenie ewentualnych możliwości kontroli błędów użytkownika przy wprowadzaniu argumentów (nieprawidłowy znak arytmetyczny/błędna dana podana przez użytkownika)

3. **Wykonanie zadania**
   Ze względu na fakt, iż nie miałem wcześniej kontaktu z językiem assembly stosunkowo ciężko było mi nie pogubić się w gąszczu nowej nomenklatury oraz szeregu „ograniczeń” jakie nakłada na użytkowniku praca na rejestrach.

   W związku z tym, oraz szeregiem nieznanych mi i niezrozumiałych błędów podczas „kompilacji”, udało mi się wykonać następujące elementy zadania (elementy zamieszczone w kodzie programu):
   - program pobiera argumenty z wiersza poleceń oraz wypisuje je na ekranie
   - program przepisuje podane argumenty do odpowiednich tablic (pomijając znak, do którego odwołuję się bezpośrednio do stosu);
   - program porównuje i wybiera odpowiednie działanie arytmetyczne
   - program ma możliwość sprawdzenia czy został podany właściwy znak arytmetyczny
   - program wykonuje prawidłowo dodawanie i odejmowanie
   - program wykonuje prawidłowo mnożenie pierwszej liczby przez **jedną cyfrę/literę** drugiej liczby podanej przez użytkownika – algorytm działa dla cyfr w systemie hexadecymalnym od 0 do 9 oraz dla litery a;
   przy literach B-F dochodzi do zjawiska gubienia przeniesienia
   - rozpoczęto implementację algorytmu dzielenia, w obecnej chwili w programie znajduje się jedynie sprawdzenie czy 2. liczba podana przez użytkownika nie jest zerem oraz wykonane zostaje „dzielenie” tylko gdy długość 1. liczby jest mniejsza od długości liczby nr 2.
   - program prawidłowo wykonuje konwersję ASCII do Hex oraz konwersję odwrotną Hex do ASCII

4. **Algorytmy oraz ich opis**

   a) pobieranie argumentów z wiersza poleceń:
   - krótki opis

   ./nazwa_programu arg1 arg2 arg3
   Argumenty, z którymi wywołany jest nasz program są umieszczane na stacku, aby się do nich dostać korzystam z możliwości jakie dają wskaźniki stosu (EBP oraz ESP).
   Poprzez odpowiednie adresowanie mam możliwość pobrać argumenty, umieścić ich wartości w rejestrach, by następnie skorzystać z innych

możliwości operacji na argumentach.
W przypadku naszego programu mamy 3 argumenty:
Arg 1 to liczba nr 1
Arg 2 to znak arytmetyczny określający jakie działanie chcemy wykonać
Arg 3 to liczba nr 2

b) pobieranie długości string
Ponieważ działania arytmetyczne chciałem przeprowadzić w sposób „bajt po bajcie" z tablic chciałem wiedzieć ile liczb ma każdy string (żeby poprawnie wpisać je do tablic – w sposób jaki przygotowałem wg algorytmów)
- opis algorytmu:
> wyczyść rejestr ebx (będzie potrzebny jako wskaźnik)
> porównaj każdy bajt stringa od miejsca wskazywanego przez [eax+ebx] (w eax mamy de facto offset wpisanego stringa) i sprawdź czy nie jest równy 0 (NULL/koniec stringa)
>jeżeli nie to wróć do początku po inkrementacji wskaźnika
>jeżeli tak przerzuć długość stringa liczby do pamięci

Po pobraniu długości każdej liczby i wprowadzeniu ich do odpowiednich „zmiennych" - strlen1 dla liczby 1 i strlen2 dla liczby 2 – wypisujemy je na ekranie terminala.

c) Wpisywanie stringa do tablicy
Aby ułatwić sobie operacje na stringach wpisuję obie liczby do odpowiadających im tablic Num1_arr oraz Num2_arr przesuwając je o 1 półbajt w prawo, by mieć na początkach tablic „zero wiodące", które przyda się w późniejszym etapie.

d) Wypisanie znaku i sprawdzenie go
Program działa na zasadzie: Przygotuj arg 1, sprawdź arg 2, przygotuj arg 3, dlatego w kodzie, mimo iż wspomniałem wyżej o przepisywaniu do tablic i wyszukiwaniu długości obu liczb pomiędzy nimi znajduje się sprawdzenie oraz wypisanie znaku podanego przez użytkownika.
Program rozpoznaje następujące znaki:
- „+" dla dodawania
- „-" dla odejmowania
- „ '*'" dla mnożenia (podawane w terminalu jako '*' ze względu na OS)
- „ '/'" dla dzielenia (podawane w terminalu jako '/' ze względu na OS)
W przypadku podania znaku, który nie został wyżej wymieniony wyświetli się informacja o wpisaniu błędnego znaku.
Po porównaniu zostaje wypisany odpowiedni znak algebraiczny i następuje przejście do wypisania i opracowania liczby 2.

e) konwersja ASCII do HEX i HEX do ASCII

Po wpisaniu liczby do tablicy program konwertuje liczbę wpisaną przez użytkownika do systemu hexadecymalnego.
Odbywa się to w następujący sposób:
1. Ustawiamy wskaźnik na tablicę z liczbą oraz ustawiamy nasz licznik na długość liczby w tablicy
2. Sprawdzamy półbajt po półbajcie liczby hexadecymalnej czym jest – cyfrą, dużą literą, małą literą.
3. Za pomocą odpowiedniej funkcji, bazując na tablicy kodów ASCII przeprowadzamy konwersję do HEX i nadpisujemy ten element w tablicy.

5. **Uwagi końcowe:**
   Zabrakło mi czasu na przygotowanie prawidłowe algorytmu mnożenia i dzielenia oraz optymalizację kodu.
6. **Optymalizacja:**
   Optymalizacja w moim programie jest możliwa w każdym jego aspekcie działania.

7. **Krótko o Makefile**
   >automatyzuje kompilację (asemblację) programu

Zawartość programu Makefile:

```
all:
    nasm -felf32 -o calc.o calc.asm
    ld -m elf_i386 -o calc calc.o
```

nasm -felf32 -o calc.o calc.asm

> asembler nasm

-felf32 - wymuszenie trybu 32 -bit

-o calc.o calc.asm – plik_wynikowy_asemblacji.o plik_źródłowy.asm

ld -m elf_i386 -o calc calc.o

> linker GNU

-m elf_i386 – wymuszenie emulacji i386

-o calc calc.o → linkowanie do pliku wynikowego z pliku źródłowego

8. Literatura

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

[2] https://pl.wikibooks.org/wiki/Asembler_x86/Architektura, WikiBooks, książka o architekturze i instrukcjach assembly dla procesorów x86

[3] „NASM – The Netwide Assembly version 2.12.02”, The NASM Development Team, 1996-2016

[4] „Język asembler dla każdego”, Bogdan Drozdowski

[5] Youtube i wszelkiej tematyki filmiki instruktażowe/wykłady związane z assembly

[6] StackOverflow – przykłady użycia instrukcji w różnych elementach programowania w języku assembly

[7] Dokumentacja instrukcji Assembly x86 dla procesorów Intela