Politechnika Wrocławska

Wydział Informatyki i Telekomunikacji

Wprowadzenie do wysokowydajnych komputerów
Sprawozdanie laboratoryjne

Obsługa wejścia/wyjścia i obliczeń arytmetycznych w języku assemblera

Wiktor Idzik - 272938

Prowadzący – mgr inż. Przemysław Świercz

18 stycznia 2025

Cel ćwiczenia

Celem ćwiczenia było napisanie programu w języku asemblera, który pobiera od użytkownika wartość zmiennej n, a następnie oblicza wartość wyrażenia:

$$n^3 + 3n^2 - 2n$$

i wyświetla wynik na ekranie. Program miał obsługiwać wyłącznie liczby całkowite, co wymagało implementacji walidacji danych wejściowych i wyświetlania komunikatu o błędzie w przypadku niepoprawnego formatu.

Dodatkowym wymaganiem było zapewnienie ochrony przed przepełnieniem (overflow), ponieważ wynik obliczeń mógł przekroczyć pojemność dostępnych rejestrów. W celu uproszczenia obsługi wejścia i wyjścia należało wykorzystać funkcje biblioteki libc.

Realizacja

Wszystkie wymagane funkcjonalności zostały zaimplementowane:

- Program poprawnie pobiera wartość n od użytkownika.
- Wykonywana jest walidacja danych wejściowych – jeśli użytkownik wprowadzi niepoprawną wartość, program wyświetla komunikat o błędzie i kończy działanie.
- Obliczenia są przeprowadzane zgodnie z podanym wzorem, a wynik jest wyświetlany na ekranie.
- Zaimplementowano mechanizm wykrywania przepełnienia, wykorzystując flagę OF (Overflow Flag) oraz instrukcję JO do obsługi błędów. W przypadku wykrycia overflow program wyświetla odpowiedni komunikat i kończy działanie.
- Do obsługi wejścia i wyjścia zastosowano funkcje scanf i printf z biblioteki libc, co uprościło implementację interakcji z użytkownikiem.

Dzięki powyższym rozwiązaniom program spełnia założenia zadania i jest odporny na błędne dane wejściowe oraz przepełnienie wartości obliczeń.

Realizacja w kodzie

Na początku programu stos jest przygotowywany: zapisujemy wskaźnik rbp na stos i ustawiamy rsp. Następnie za pomocą funkcji printf wyświetlany jest komunikat "Podaj n: ", który znajduje się w zmiennej message. Argumenty dla printf są przekazywane przez rejestry, gdzie rdi przechowuje adres tekstu, a rax jest ustawione na 0.

```
main:
    # Przygotowanie stosu
    pushq %rbp
    movq %rsp, %rbp

    # Wyswietlenie w konsoli "Podaj n: "
    leaq message(%rip), %rdi
    movq $0, %rax
    call printf
```

W tym fragmencie program wywołuje funkcję scanf, aby pobrać liczbę całkowitą od użytkownika. Format wejściowy %d jest przekazywany przez rejestr rdi, a adres zmiennej num przez rsi. Funkcja scanf zapisuje wprowadzone dane do zmiennej num. Ten fragment kodu jest klasycznym prologiem funkcji w asemblerze zgodnym z konwencją System V ABI. Jest to standardowe podejście, które zabezpiecza poprzednią ramkę stosu i ustawia %rbp jako wskaźnik na nową ramkę funkcji.

```
# Pobranie n od użytkownika
leaq input(%rip), %rdi
leaq num(%rip), %rsi  # Adres zmiennej num
movq $0, %rax
call scanf
```

Po wywołaniu scanf sprawdzamy, czy dane zostały poprawnie wczytane. Funkcja scanf zwraca liczbę poprawnie wczytanych argumentów w rejestrze eax. Jeśli wartość jest różna od 1, program przechodzi do error_handler i wyświetla komunikat o błędzie.

```
# Sprawdzenie, czy dane poprawne
cmpl $1, %eax
jne error_handler
```

Instrukcja jo (jump if overflow) sprawdza ustawienie flagi przepełnienia po mnożeniu. Jeśli wystąpiło przepełnienie, przesyłamy sterowanie do overflow_handler. Flaga przepełnienia jest sprawdzana przy każdej operacji arytmetycznej.

```
# Obliczanie n^3
imull %eax, %eax
jo overflow_handler        # Sprawdzenie overflow
```

Wynik jest wyświetlany na ekranie za pomocą funkcji printf. Zmienna output zawiera format tekstu do wypisania, a wynik jest przekazywany do funkcji w rejestrze esi.

```
# Wypisanie wyniku
leaq output(%rip), %rdi
movl %eax, %esi
movq $0, %rax
call printf
```

Program kończy swoje działanie, przywracając stos i kończąc funkcję main.

```
# Zakonczenie programu
movq $0, %rax
movq %rbp, %rsp
popq %rbp
ret
```

Jeśli dane wejściowe są niepoprawne, program przechodzi do error_handler, gdzie wyświetlany jest komunikat o błędzie, a program kończy się z kodem błędu 1.

```
error_handler:
    # Wypisanie komunikatu o bledzie
    leaq error(%rip), %rdi
    movq $0, %rax
    call printf

    # Zakonczenie programu z kodem bledu 1
    movq $1, %rax
    movq %rbp, %rsp
    popq %rbp
    ret
```

Jeśli w trakcie obliczeń nastąpi przepełnienie, program przechodzi do overflow_handler, gdzie wyświetlany jest komunikat o błędzie przepełnienia, a program kończy się z kodem błędu 2.

```
overflow_handler:
    # Wypisanie komunikatu o bledzie przepelnienia
    leaq overflow_error(%rip), %rdi
    movq $0, %rax
    call printf

    # Zakonczenie programu z kodem bledu 2
    movq $2, %rax
    movq %rbp, %rsp
    popq %rbp
    ret
```

Źródła:

- https://uclibc.org/docs/psABI-x86_64.pdf - Definiuje konwencję wywołań funkcji, rejestry oraz sposób przekazywania argumentów do printf i scanf.
- https://stackoverflow.com/questions/14523480/assembly-detecting-overflow-register - Metody sprawdzania przepełnienia.
- IA-32 Intel Architecture Software Developers Manual vol. 1 - Basic Architecture.pdf – Podstawy budowy procesora Intel
- IA-32 Intel Architecture Software Developers Manual vol. 2 - Instruction Set Reference.pdf – Podstawy języka asembler x86
- https://gcc.gnu.org/onlinedocs/ - Podstawy kompilacji za pomocą gcc
- https://man7.org/linux/man-pages/man3/scanf.3.html - Dokumentacja funkcji scanf