Architektura Komputerów 2 – Laboratoria
Laboratorium nr 2
Piotr Stachnio 241268

Cel laboratoriów

Celem laboratoriów było stworzenie programów w jeżyku assembly dla architektury x86_32 które wykonywały by podstawowe operacja na wielkich liczbach. Za wielkie liczby uznajemy te które są większe od pojedyńczego słowa czyli w przypadku x86_32 większe od 32 bitów.

Wstęp

Jako linkera zdecydowałem się na użycie GNU linkera, a do asemblacji GNU assemblera.

Zadanie nr1

Głównym elementem programu jest pętla w której wykonuję dodawanie z uwzględnieniem przeniesienia na następną pozycję za pomocą rozkazu **adcl,** wszystkie dane potrzebne do dodania dwóch wielkich liczb są przechowywane we wcześniej zadeklarowanej strukturze danych **.long** która, przynajmniej mi, przypomina array. Po wykonaniu pętli sprawdzam czy flaga cf, odpowiadająca za sygnalizowanie przeniesienia, jest równa jeden. Jeżeli tak pushuje na stos 1, w przeciwnym przypadku następuje zakończenie działania programu.

```
SYS_EXIT = 1
EXIT_SUCCESS = 0

.code32

.data
liczba1:
        .long 0x10304008, 0x701100FF, 0x45100020, 0x08570030
liczba2:
        .long 0xF040500C, 0x00220026, 0x321000CB, 0x04520031

.text
.global _start

_start:

    movl $4, %ecx    # licznik petli, musi byc w ecx poniewaz wracam do _loopo_start za pomoca loop
    clc                # resetowanie flagi carry do wartosci 0
_loop_start:
    movl liczba1(, %ecx, 4), %eax # skopiowanie pojedynczgo slowa liczby 1 do eax
    movl liczba2(, %ecx, 4), %ebx # skopiowanie pojedynczego slowa liczby 2 do ebx

    decl %ecx # dekrementujemy nasz iterator
    adcl %eax, %ebx # dodajemy z przeniesieniem do eax ebx
    pushl %ebx # wynik zapisuje na stosie
    loop _loop_start

jnc _bez_przeniesienia # jezeli cf = 0 idziemy do _bez_przeniesienia, w przeciwnym wypadku to pushujemy na stos 1
    pushl $1
    jmp _end # skok do end
_bez_przeniesienia:
    pushl $0
    jmp _end
_end:
    mov $SYS_EXIT, %eax
    mov $EXIT_SUCCESS, %ebx
    int $0x80  # wywolanie funkcji systemowej o parametrach SYS_EXIT, EXIT_SUCCESS, jest to funkcja odpowiadająca za kończenie programu

```

Zadanie nr2

Zadanie nr 2 polegało na napisaniu programu który realizowałby operację odejmowania. Program ten jest analogiczny do tego z zadanie pierwszego różnica polegała na użyciu innego rozkazu w pętli. Zamiast **adcl sbbl** który wykonuje operację usunięcia z uwzględnieniem tzw. pożyczki

```
SYS_EXIT = 1
EXIT_SUCCESS = 0

.code32

.data
liczba1:
        .long 0x10304008, 0x701100FF, 0x45100020, 0x08570030
liczba2:
        .long 0xF040500C, 0x00220026, 0x321000CB, 0x04520031

.text
.global _start

_start:

    movl $4, %ecx    # licznik petli, musi byc w ecx poniewaz wracam do _loopo_start za pomoca loop
    clc                 # resetowanie flagi carry do wartosci 0
_loop_start:
    movl liczba1(, %ecx, 4), %eax # skopiowanie pojedynczgo slowa liczby 1 do eax
    movl liczba2(, %ecx, 4), %ebx # skopiowanie pojedynczego slowa liczby 2 do ebx

    decl %ecx # dekrementujemy nasz iterator
    sbbl %eax, %ebx # odejmujemy z "porzyczka"
    pushl %ebx # wynik zapisuje na stosie
    loop _loop_start


_end:
    mov $SYS_EXIT, %eax
    mov $EXIT_SUCCESS, %ebx
    int $0x80  # wywolanie funkcji systemowej o parametrach SYS_EXIT, EXIT_SUCCESS

```

MakeFile

Aby uniknąć wpisywania kommend typu: as adder.s -o adder.o stworzyłem plik makefile który jest odpowiedzialny za asemblację oraz linkowanie stworzonych przezemnie programów. Makefile jest swego rodzaju skryptem w który wykonuje komendy shellowe. Stworzony przezemnie makefile posiadał również operację clean która usuwała pliki tworzone w przez assembler oraz linker.

```
all: adder subtractor


adder: adder.s
    as adder.s -o adder.o
    ld adder.o -o adder

clean:
    rm -rf adder.o adder
    rm -rf subtractor.o subtractor

subtractor: subtractor.s
    as subtractor.s -o subtractor.o
    ld subtractor.o -o subtractor
```

Problemy z realizacją

Podczas realizacji zadań napotkałem wiele problemów z którymi musiałem się zmierzyć. Jednym z nich było wybieranie odpowiedniego słowa wchodzącego w skład jednej z liczb. Również problem sprawiało przestawienie myślenia które jest znanie z programowania wysoko poziomowego na programowanie na bardzo potężnym ale jednak ciągle układzie cyfrowym.

Podsumowanie

Realizowane zadanie były wymagającę, zmuszające do zmiany sposobu w który patrzyłem na kod. Podczas pisania programów byłme zmuszony wielokrotnie do szukania informacji w takich źródłach jak:
https://sourceware.org/binutils/docs-2.22/as/index.html
https://chromium.googlesource.com/chromiumos/docs/+/master/constants/syscalls.md#x86-32_bit

Nie udało mi się zrealizować mnożenie wielkich liczb.