Wrocław, 18 marca 2018

Marcin Kotas, 235098 WT-P-7.15

prowadząca: mgr Aleksandra Postawka

Laboratorium Architektury Komputerów
(2) Utrwalenie umiejętności tworzenia prostych konstrukcji programowych, obsługa plików

1 Treść ćwiczenia

Zakres i program ćwiczenia:

Utworzenie programu wczytującego dwie liczby zapisane w kodzie szesnastkowym z plików i dodającego je z użyciem flagi przeniesienia. Wynik zapisywany w osobnym pliku w kodzie czwórkowym.

2 Kod programu

2.1 Sekcja danych

W sekcji danych umieszczone zostały dodatkowe nazwy symboliczne użyte do obsługi plików. Wartości `SYSOPEN` oraz `SYSCLOSE` odnoszą się do funkcji otwierania oraz zamykania plików. Wartość `O_WR_CRT_TRNC` została wyliczona jako suma wartości dla tylko odczytu (01), tworzenia pliku jeśli nie ma (0100) oraz nadpisywania poprzednich wartości pliku (01000). Zmienne `f_***` zawierają nazwy plików używanych w programie.

```
.data
    SYSEXIT = 60
    EXIT_SUCCESS = 0
    SYSREAD = 0
    SYSWRITE = 1
    STDOUT = 1
    SYSOPEN = 2
    SYSCLOSE = 3
    O_RDONLY = 00
    O_WRONLY = 01
    O_WR_CRT_TRNC = 01101

    BUFLEN = 1024

    f_in1: .ascii "in1\0"
    f_in2: .ascii "in2\0"
    f_out: .ascii "out\0"

.bss
    .comm num1_buf, 1024
    .comm num1, 1024
    .comm num2_buf, 1024
    .comm num2, 1024
    .comm sum_buf, 1024
    .comm sum_out, 1024
```

2.2 Wczytywanie liczby z pliku

Otwieranie pliku odbywa się za pomocą funkcji `SYSOPEN`. Pierwszym argumentem tej funkcji jest nazwa pliku. Drugim jest tryb otwarcia, a ostatnim kod dostępu.
Następnie przy pomocy funkcji `SYSREAD` zawartość pliku wczytywana jest do bufora.
Funkcja `SYSCLOSE` zamyka plik. Jej jedynym argumentem jest deskryptor pliku.

```
num1_file:
# %r14 - number of num1 bytes
    movq $SYSOPEN, %rax
    movq $f_in1, %rdi
    movq $O_RDONLY, %rsi
    movq $0666, %rdx
    syscall

    movq %rax, %rdi
# file handle
    movq $SYSREAD, %rax
    movq $num1_buf, %rsi
    movq $BUFLEN, %rdx
    syscall

    movq %rax, %r14
    sub $3, %r14
# -'\n', taking 2 bytes at a time

    movq $SYSCLOSE, %rax
# file handle still in %rdi
    syscall
```

2.3 Konwersja znaków w buforze na liczbę

Pętla `read_num1` sczytuje z bufora od końca po 2 bajtach, następnie sprawdza, czy oba bajty zawierają cyfrę w kodzie ascii. Służy do tego funkcja `to_number`, która sprawdza znak i zwraca cyfrę w rejestrze `al` oraz informację, czy znak był cyfrą w rejestrze `ah`.
Następnie obie cyfry zapisywane są w jednym bajcie w buforze num1. Po sczytaniu wszystkich par bajtów następuje sprawdzenie, czy liczba bajtów była nieparzysta. Jeśli tak, następuje wczytanie bajtu o indeksie 0 i dopisanie go na koniec bufora.

```
num1_decode:
    movq %r14, %r8
    movq %r8, %rdi
    movq $0, %rsi
    read_num1:
        movw num1_buf(, %rdi, 1), %bx
        movb %bl, %al
        call to_number
        cmp $0, %ah
        jne exit

        movb %al, %bl
        shl $4, %bl

        movb %bh, %al
        call to_number
        cmp $0, %ah
        jne exit

        or %al, %bl
        movb %bl, num1(, %rsi, 1)

        inc %rsi
        sub $2, %rdi
        cmp $0, %rdi
        jge read_num1
    movq %r8, %rax
    movq $0, %rdx
    movq $2, %r8
    div %r8
    cmp $0, %rdx
    je num2_decode

    movq $0, %rdi
# if odd number
    movb num1_buf(, %rdi, 1), %al
    call to_number
    cmp $0, %ah
    jne exit
    movb %al, num1(, %rsi, 1)
```

2.4 Dodanie liczb

Najpierw dodawane jest bez przeniesienia ostatnie 16 cyfr liczb. Rejestr flag kopiowany jest na stos (`pushf`), ponieważ funkcja `cmp` nadpisuje flagę przeniesienia. Następnie w pętli `add_carry` wynik dodawania kopiowany jest do bufora `sum_buf`, rejestr flag zostaje przywrócony z kopca (`popf`) i dodane jest z przeniesieniem kolejne 8 bajtów liczb. Po wykonaniu dodawania rejestr flagowy znów zostaje wrzucony na stos. Pętla wykonuje się dopóki wynik dodawania jest większy od zera.

```
add_no_carry:
    movq $0, %rdi
    movq num1(, %rdi, 8), %rax
    movq num2(, %rdi, 8), %rbx
    add %rbx, %rax
    pushf

    add_carry:
        movq %rax, sum_buf(, %rdi, 8)
        inc %rdi
        popf
        movq num1(, %rdi, 8), %rax
        movq num2(, %rdi, 8), %rbx
        adc %rbx, %rax
        pushf
        cmp $0, %rax
        jne add_carry
    popf
    adc $0, %rax
# add carry, if %rax=ff..f+1
    cmp $0, %rax
    je no_carry
    movb $1, sum_buf(, %rdi, 8)
    inc %rdi
```

2.5 Konwersja na kod czwórkowy

Do rejestru `r11` zapisana jest liczba bajtów w buforze `sum_buf`. Jest to indeks z pętli `add_carry` pomnożony przez 8.
W pętli `to_stack` brane są kolejne 2 bity z bufora (poprzez wykonanie funkcji and z liczbą 3), do których dodawany jest kod ascii '0' i wrzucane są na stos. Następnie rejestr `al` przesuwany jest o 2 miejsca, aby mogły zostać pobrane kolejne 2 bity.
Pętla `to_quater` wykonuje się tak długo, aż odczytane zostaną wszystkie bajty z bufora (liczba bajtów w rejestrze `r11`).
Po wykonaniu się pętli `to_quater` następuje sczytanie kolejnych znaków z rejestru do bufora i wypisanie do pliku. Otwarcie pliku następuje przy użyciu tych samych funkcji co na początku, tym razem z użyciem parametru `O_WR_CRT_TRNC` zamiast `O_RDONLY`.
Zapisanie bufora do pliku odbywa się za pomocą funkcji `SYSWRITE`, gdzie jako pierwszy parametr podawany jest deskryptor otwartego pliku.

```
no_carry:
    movq %rdi, %rax
    movq $8, %r8
    mul %r8
    movq %rax, %r11
# %r11 - number of sum_buf bytes
    movq $0, %rdi
    movq $0, %r9
# to_stack iterator
    to_quater:
        movb sum_buf(, %rdi, 1), %al
        inc %rdi
        to_stack:
            movb %al, %bl
            and $3, %bl
            shr $2, %al
            add $'0', %bl
            push %rbx
            inc %r9
            cmp $4, %r9
            jl to_stack

        movq $0, %r9
        cmp %r11, %rdi
        jl to_quater
```

3 Wnioski

Program uruchomił się poprawnie. Przy otwieraniu plików można podać jako parametr kod dostępu 0666 (ósemkowo), który nadaje wszystkim użytkownikom pełną kontrolę nad plikiem. Przy zapisywaniu liczb do pamięci należy pamiętać o konwencji little endian, gdzie liczby należy zapisywać od najmłodszego bitu.