Wrocław, 3 marca 2018

Marcin Kotas, 235098 WT-P-7.15

prowadząca: Aleksandra Postawka

Laboratorium Architektury Komputerów
(0) Podstawy uruchamiania programów asemblerowych na platformie Linux

1 Treść ćwiczenia

Zakres i program ćwiczenia:

Techniki tworzenia i uruchamiania programów napisanych w języku asemblera AT&T. Obsługa debugger'a gdb.

Zrealizowane zadania:

Utworzenie prostego programu „Hello, world" oraz pliku Makefile w celu zautomatyzowania kompilacji i konsolidacji programu za pomocą komendy make.

Napisanie programu wczytującego tekst ze standardowego wejścia, zamieniającego wielkie litery na małe i na odwrót.

2 Przebieg ćwiczenia

2.1 Program „Hello, world!"

Na początku programu znajduje się sekcja danych (.data). Zawiera ona definicje nazw symbolicznych oraz deklaracje zmiennych i buforów. Sekcja .text zawiera zapis algorytmu programu. _start oznacza wejście. Funkcja movq kopiuje do kolejnych rejestrów nazwę a następnie argumenty funkcji. Dyrektywa syscall wywołuje funkcję. Funkcja SYSWRITE służy do wydrukowania napisu, a funkcja SYSEXIT wychodzi z programu.

```
.data
    SYSREAD = 0
    SYSWRITE = 1
    SYSEXIT = 60
    STDOUT = 1
    STDIN = 0
    EXIT_SUCCESS = 0

    buf: .ascii "Hello, world!\n"
    buf_len= .-buf

.text
.globl _start
_start:
    movq $SYSWRITE, %rax
    movq $STDOUT, %rdi
    movq $buf, %rsi
    movq $buf_len, %rdx
    syscall

    movq $SYSEXIT, %rax
    movq $EXIT_SUCCESS, %rdi
    syscall
```

2.2 Kompilacja programu

Plik należy najpierw skompilować do obiektu za pomocą funkcji `as -o plik.o plik.s`. Następnie plik.o należy skonsolidować przy użyciu funkcji `ld -o plik plik.o`. W celu zapamiętania konfiguracji kompilacji warto stworzyć plik Makefile, w którym zachowane będą kolejne instrukcje:

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

Aby wykonać kompilację za pomocą tego pliku należy użyć w terminalu komendy `make`. Aby uruchomić program należy użyć komendy `./plik` będąc w folderze z programem.

2.3 Debugowanie programu

Aby przeprowadzić debugowanie programu za pomocą gdb należy skompilować program z odpowiednimi opcjami, które powiążą polecenia i etykiety z końcowymi kodami. Można to zrobić kompilując program za pomocą kompilatora gcc: `gcc -g -o plik plik.s`, przy czym należy pamiętać, by w kodzie programu zamienić `.globl` na `.global` oraz `_start` na `main`.

Drugim sposobem na kompilację programu z obsługą debugowania jest dodanie flagi `-gstabs` przy wywoływaniu kompilatora `as`. Nie trzeba wtedy zmieniać nazw w kodzie, jednak w starszych wersjach debugera istniała potrzeba dodania funkcji `nop` na początek programu ponieważ nie dało się ustawić breakpoint'a na etykiecie _start.

Gdb umożliwia dokładne śledzenie działania programu poprzez zatrzymywanie wykonywania w konkretnym miejscu, a następnie śledzenie krok po kroku i monitorowanie zawartości rejestrów. Aby ustawić breakpoint na konkretnej instrukcji należy wpisać komendę `b [adres]`. Można posłużyć się etykietami: `b *_start`. Do uruchomienia programu służy komenda `r`. Po zatrzymaniu programu można wykonywać instrukcje krok po kroku (`s`) lub kontynuować normalną pracę programu (`c`). W każdej chwili pożna wyświetlić zawartość wszystkich rejestrów (`info registers`) lub poszczególnych (`p/d $<nazwa rejestru>`). Można również wyświetlić zawartość zmiennych odnosząc się do nich za pomocą `&<nazwa zmiennej>`.

2.4 Program zamieniający wielkość liter

W sekcji data dodana została zmienna BUFLEN, która definiuje długość w bajtach bufora, do którego wczytywany będzie tekst.
Sekcja .bss zawiera niezainicjalizowane dane. Tam znajdują się bufory, do których zostanie zapisany wprowadzony tekst.

```
.data
    SYSREAD = 0
    SYSWRITE = 1
    SYSEXIT = 60
    STDOUT = 1
    STDIN = 0
    EXIT_SUCCESS = 0
    BUFLEN = 512

.bss
    .comm buf_in, 512
    .comm buf_out, 512
```

Do wczytania tekstu z klawiatury wykorzystana jest funkcja `SYSREAD`, a jako argument podany jest standardowy strumień wejściowy `STDIN`. Wprowadzony tekst zapisywany jest do bufora `buf_in`. Następnie zawartość rejestru `rax` jest zmniejszana o 1, aby uwzględnić znak końca linii.

W pętli `zamien_litery` zawartość bufora jest kopiowana po jednym bajcie do rejestru `bh`. Poprzez porównanie z kodami liter za pomocą funkcji `cmp`, znak w rejestrze klasyfikowany jest jako litera lub inny znak. Jeżeli znak jest literą, to poprzez funkcję `xor` zamieniana jest wielkość litery (liczba `0x20` posiada tylko jeden bit 1 na miejscu, w którym różnią się liczby wielkie od małych). Na końcu pętli aktualny znak jest kopiowany do bufora wyjściowego, po czym porównywana jest wartość licznika (rejestr `rdi`) z ilością znaków w buforze (rejestr `rax`). W tym przypadku wykorzystywany jest skok warunkowy `jl` - jump if less.

Po wykonaniu się pętli do bufora dodawany jest znak końca linii, po czym funkcja SYSWRITE drukuje do strumienia `STDOUT` zawartość bufora `buf_out`. Na końcu wykonuje się funkcja SYSEXIT.

```
.text
.globl _start

_start:
    movq $SYSREAD, %rax
    movq $STDIN, %rdi
    movq $buf_in, %rsi
    movq $BUFLEN, %rdx
    syscall

    dec %rax    #'\n'
    movq $0, %rdi #licznik

    zamien_litery:
        movb buf_in(,%rdi, 1), %bh
        cmp $'A', %bh
        jl nie_litera

        cmp $'Z', %bh
        jle litera

        cmp $'a', %bh
        jl nie_litera

        cmp $'z', %bh
        jg nie_litera

        litera:
        movb $0x20, %bl
        xor %bl, %bh

        nie_litera:
        movb %bh, buf_out(, %rdi, 1)
        inc %rdi
        cmp %rax, %rdi
        jl zamien_litery

    movb $'\n', buf_out(, %rdi, 1)

    movq $SYSWRITE, %rax
    movq $STDOUT, %rdi
    movq $buf_out, %rsi
    movq $BUFLEN, %rdx
    syscall

    movq $SYSEXIT, %rax
    movq $EXIT_SUCCESS, %rdi
    syscall
```

3 Wnioski

Programy uruchomiły się poprawnie. W nowszych wersjach gdb nie ma już potrzeby wstawiania funkcji no operation po _start, jeśli używa się kompilatora asemblerowego (as). Można również kompilować używając kompilatora c (gcc), który sprawdza się równie dobrze.