Федеральное государственное автономное образовательное учреждение высшего образования
Университет ИТМО

Кафедра Вычислительной Техники

Дисциплина: Низкоуровневое программирование

Лабораторная работа №2

Выполнил: **Доморацкий Эридан Алексеевич**

Группа: **P33113**

Преподаватель: **Логинов Иван Павлович**

2020г

Задание

5.4 Assignment: Dictionary

This assignment will further advance us to a working Forth interpreter. Some things about it might seem forced, like the macro design, but it will make a good foundation for an interpreter we are going to do later.

Our task is to implement a dictionary. It will provide a correspondence between keys and values. Each entry contains the address of the next entry, a key, and a value. Keys and values in our case are null-terminated strings.

The dictionary entries form a data structure are called a **linked list**. An empty list is represented by a null pointer, equal to zero. A non-empty list is a pointer to its first element. Each element holds some kind of value and a pointer to the next element (or zero, if it is the last element).

Выполнение

```
; colon.inc
; vim: syntax=nasm

%define colon_prev 0

%macro colon 2
%2:
   dq colon_prev
   dq %%value
   db %1, 0
%%value:
%define colon_prev %2
%endmacro

; dict.asm
; vim: syntax=nasm

global find_word

extern string_equals

; Find word in dict
; rdi - key, rsi - dict
; rax - entry address
find_word:
.loop:                  ; do {
   push rdi
   push rsi
   add rsi, 16
   call string_equals  ;     rax = string_equals(key, dict.key);
   pop rsi
   pop rdi

   test rax, 1         ;     if (rax & 1 != 0)
   jnz .succ           ;         goto .succ;

   mov rsi, [rsi]      ;     dict = dict.prev;
   test rsi, rsi       ; } while (dict != 0);
   jnz .loop

   xor rax, rax        ; return 0;
   ret

.succ:
   mov rax, rsi        ; return dict;
   ret
```

```
; lib.asm
; vim: syntax=nasm

global exit
global string_length
global print_string
global print_error
global print_char
global print_newline
global print_uint
global print_int
global string_equals
global read_char
global read_word
global parse_uint
global parse_int
global string_copy


section .data
rsp_buffer: dq 0

section .text

; Выполняет syscall с сохранением регистров
do_syscall:
   push rcx
   mov [rsp_buffer], rsp

   syscall

   mov rsp, [rsp_buffer]
   pop rcx
   ret


; Принимает код возврата и завершает текущий процесс
exit:
   mov rax, 60
   syscall
   ret


; Принимает указатель на нуль-терминированную строку, возвращает её длину
string_length:
   xor rax, rax            ; int count = 0

.loop:
   cmp byte [rdi + rax], 0 ; while (str[count] != 0) {
   je .end

   inc rax                 ;    ++count;
   jmp .loop               ; }

.end:
   ret                     ; return count;


; Принимает указатель на нуль-терминированную строку, выводит её в stdout
print_string:
   push rax
   push rdi
   push rsi
   push rdx

   call string_length
   mov rdx, rax
   mov rax, 1
   mov rsi, rdi
   mov rdi, 1
   call do_syscall     ; write(stdout, str, string_length(str));

   pop rdx
   pop rsi
   pop rdi
   pop rax
   ret
```

```
print_error:
   push rax
   push rdi
   push rsi
   push rdx

   call string_length
   mov rdx, rax
   mov rax, 1
   mov rsi, rdi
   mov rdi, 2
   call do_syscall     ; write(stderr, str, string_length(str));

   pop rdx
   pop rsi
   pop rdi
   pop rax
   ret

; Принимает код символа и выводит его в stdout
print_char:
   push rax
   push rdi
   push rsi
   push rdx

   dec rsp
   mov byte [rsp], dil ; char buffer = c;
   mov rax, 1
   mov rdi, 1
   mov rsi, rsp
   mov rdx, 1
   call do_syscall     ; write(stdout, &buffer, 1);
   inc rsp

   pop rdx
   pop rsi
   pop rdi
   pop rax
   ret


; Переводит строку (выводит символ с кодом 0xA)
print_newline:
   push rdi

   mov dil, 0x0a
   call print_char ; print_char('\n');

   pop rdi
   ret


; Выводит беззнаковое 8-байтовое число в десятичном формате
; Совет: выделите место в стеке и храните там результаты деления
; Не забудьте перевести цифры в их ASCII коды.
print_uint:
   push rax
   push rbx
   push rcx
   push rdx
   push rdi
   xor rcx, rcx
   mov rax, rdi

   dec rsp
   mov byte [rsp], 0   ; stack.push_front(0);

   test rax, rax
   je .loop_body

.loop:
   test rax, rax
   je .end

.loop_body:
   xor rdx, rdx
```

```
   mov rbx, 10
   div rbx             ; rax := rax / 10; dl := rax % 10;
   add dl, '0'         ; dl := digitToChar(dl);
   dec rsp
   mov byte [rsp], dl  ; stack.push_front(dl);
   inc cx              ; ++cx;
   jmp .loop

.end:
   mov rdi, rsp
   call print_string   ; print_string(stack);
   add rsp, rcx        ; stack.size() -= cx;

   inc rsp
   pop rdi
   pop rdx
   pop rcx
   pop rbx
   pop rax
   ret


; Выводит знаковое 8-байтовое число в десятичном формате
print_int:
   push rdi

   cmp rdi, 0
   jge .print      ; if (rdi < 0) {

   push rdi
   mov rdi, '-'
   call print_char ;    print_char('-');
   pop rdi
   neg rdi         ;    rdi = -rdi;

.print:             ; }
   call print_uint ; print_uint(rdi);

   pop rdi
   ret


; Принимает два указателя на нуль-терминированные строки, возвращает 1 если они равны, 0 иначе
string_equals:
   push rbx

   xor rax, rax
.loop:
   mov bl, byte [rsi + rax]
   cmp byte [rdi + rax], bl
   jne .fail

   cmp byte [rdi + rax], 0
   je .succ

   inc rax
   jmp .loop

.fail:
   xor rax, rax
   jmp .end

.succ:
   mov rax, 1

.end:
   pop rbx
   ret


; Читает один символ из stdin и возвращает его. Возвращает 0 если достигнут конец потока
read_char:
   push rdi
   push rsi
   push rdx

   dec rsp
   xor rax, rax
```

```
   mov rdi, 0
   mov rsi, rsp
   mov rdx, 1
   call do_syscall

   test rax, rax
   je .stream_end

   xor rax, rax
   mov al, byte [rsp]
   jmp .end

.stream_end:
   xor rax, rax

.end:
   inc rsp
   pop rdx
   pop rsi
   pop rdi
   ret


; Принимает: адрес начала буфера, размер буфера
; Читает в буфер слово из stdin, пропуская пробельные символы в начале, .
; Пробельные символы это пробел 0x20, табуляция 0x9 и перевод строки 0xA.
; Останавливается и возвращает 0 если слово слишком большое для буфера
; При успехе возвращает адрес буфера в rax, длину слова в rdx.
; При неудаче возвращает 0 в rax
; Эта функция должна дописывать к слову нуль-терминатор
read_word:

.whitespaces:
   call read_char

   cmp rax, ' '
   je .whitespaces
   cmp rax, 0x9
   je .whitespaces
   cmp rax, 0xA
   je .whitespaces

   xor rdx, rdx
.loop:
   cmp rdx, rsi
   je .fail

   cmp rax, ' '
   je .succ
   cmp rax, 0x9
   je .succ
   cmp rax, 0xA
   je .succ
   test rax, rax
   je .succ

   mov byte [rdi + rdx], al
   inc rdx

   call read_char
   jmp .loop

.fail:
   xor rax, rax
   jmp .end

.succ:
   mov rax, rdi
   mov byte [rdi + rdx], 0

.end:
   ret


; Принимает указатель на строку, пытается
; прочитать из её начала беззнаковое число.
; Возвращает в rax: число, rdx : его длину в символах
; rdx = 0 если число прочитать не удалось
```

```
parse_uint:
   push rbx

   xor rdx, rdx
   xor rax, rax
   xor rbx, rbx
.loop:                           ; while (true) {
   mov bl, byte [rdi + rdx]    ;    int digit = charToDigit(str[i])
   sub bl, '0'
   jl .end                     ;    if (digit < 0) break;
   cmp bl, 9
   jg .end                     ;    if (digit > 9) break;

   push rdx
   mov rdx, 10
   mul rdx                     ;    acc *= 10;
   pop rdx
   add rax, rbx                ;    acc += digit;

   inc rdx                     ;    ++i;
   jmp .loop                   ; }

.end:
   pop rbx
   ret


; Принимает указатель на строку, пытается
; прочитать из её начала знаковое число.
; Если есть знак, пробелы между ним и числом не разрешены.
; Возвращает в rax: число, rdx : его длину в символах (включая знак, если он был)
; rdx = 0 если число прочитать не удалось
parse_int:
   push rdi

   cmp byte [rdi], '-'
   je .negative

   cmp byte [rdi], '+'
   je .positive

   call parse_uint
   jmp .end

.negative:
   inc rdi
   call parse_uint
   test rdx, rdx
   je .end

   neg rax
   inc rdx
   jmp .end

.positive:
   inc rdi
   call parse_uint
   test rdx, rdx
   je .end

   inc rdx

.end:
   pop rdi
   ret


; Принимает указатель на строку, указатель на буфер и длину буфера
; Копирует строку в буфер
; Возвращает длину строки если она умещается в буфер, иначе 0
string_copy:
   push rbx
   xor rax, rax

.loop:
   cmp rax, rdx
   je .fail
```

```
    mov bl, byte [rdi + rax]
    mov byte [rsi + rax], bl
    cmp byte [rsi + rax], 0
    je .end

    inc rax
    jmp .loop

.fail:
    xor rax, rax

.end:
    pop rbx
    ret

; main.asm
; vim: syntax=nasm

%include "colon.inc"

section .data
%include "words.inc"

value_strconst: db "Value: ", 0
too_long_key_strconst: db "Too long key!", 0x0a, 0
key_not_found_strconst: db "Key not found.", 0x0a, 0

section .text

global _start

extern read_word
extern find_word
extern print_string
extern print_newline
extern print_error
extern exit

_start:
    sub rsp, 256
    mov rdi, rsp
    mov rsi, 256
    call read_word

    test rax, rax
    je .too_long_error

    mov rdi, rax
    mov rsi, dict
    call find_word

    test rax, rax
    je .not_found_error

    mov rdi, value_strconst
    call print_string

    mov rdi, [rax + 8]
    call print_string
    call print_newline

    xor rdi, rdi
    call exit
    ret

.too_long_error:
    mov rdi, too_long_key_strconst
    call print_error

    mov rdi, 1
    call exit


.not_found_error:
    mov rdi, key_not_found_strconst
    call print_error

    mov rdi, 2
```

```
   call exit

; words.inc
; vim: syntax=nasm

colon "VK", vk
db "https://vk.com/", 0

colon "Mail.ru", mail
db "https://mail.ru/", 0

colon "YouTube", youtube
db "https://youtube.com/", 0

dict:
colon "Yandex", yandex
db "https://yandex.ru/", 0
```

Вывод

При выполнении данной лабораторной работы была освоена работа с препроцессором NASM и написание макросов на языке препроцессора.