МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра электронных вычислительных машин

Отчет по предмету
Проектирование на языке ассемблера
Вариант 3

Лабораторная работа 7
«Загрузка и выполнение программ. Работа с памятью»

Выполнил:
Студент группы 150503
Шарай П.Ю.

Проверил:
Туровец Н.О.

Минск 2022

Теоретические сведения:
Для выполнения работы требуется рассмотреть следующие элементы языка ассемблера и операционной системы:

1. Управление памятью.

При запуске программы в DOS, ей выделяется все доступная память, по-этому доступно использование, например, памяти за концом программы прак- тически до конца сегмента кода. Однако для загрузки других программ из те- кущей потребуется свободная память и наиболее простой вариант ее получения – сократить выделенный текущей программе блок памяти до минимума с по- мощью функции DOS `4Ah`.

-- Функция DOS 4Ah (`INT 21h`) – изменить размер блока памяти:

Ввод: Вывод:

`AH = 4Ah`,
`BX` =новыйразмерв16-байтныхпараграфах,
`ES =` сегментный адрес модифицируемого блока.

если `CF = 1`, то есть ошибки (в AX – код ошибки):
-- `07` – блоки управления памятью разрушены, --`08`– нехватаетпамяти(приувеличении,в `BX` =максимальный

размер, доступный для этого блока), -- `09` – `ES` содержит неверный адрес.

Если `CF = 0`, то операция выполнена успешно.

Также доступно выделение и удаление дополнительных блоков памяти: -- Функция DOS 48h (`INT 21h`) – выделить блок памяти:

Ввод: `AH = 48h`
`BX` =размерблокав16-байтныхпараграфах.
Эта функция с `BX = FFFFh` используется для определения размера самого большого доступного блока памяти.

Вывод:

если `CF = 1`, то есть ошибки (в AX – код ошибки):
-- `07` – блоки управления памятью разрушены,
-- `08` – не хватает памяти (BX = размер максимального доступного

блока).
Если CF = 0, то операция выполнена успешно, AX = сегментный адрес выделенного блока.

-- Функция DOS 49h (INT 21h) – освободить блок памяти:

Ввод: Вывод:

AH = 49h
ES = сегментный адрес освобождаемого блока

если CF = 1, то есть ошибки (в AX – код ошибки): -- 07 – блоки управления памятью разрушены,
-- 09 – ES содержит неверный адрес.

Если CF = 0, то операция выполнена успешно.

2. Загрузка и выполнение программ.

Для загрузки и выполнения программ требуется использовать функцию DOS4Bh(INT 21h)–загрузитьивыполнитьпрограмму:

Ввод:

AH = 4Bh,
AL =подфункции:
AL = 00h—загрузитьивыполнить; AL = 01h—загрузитьиневыполнять:

DS:DX – адрес ASCIZ-строки с полным именем программы, ES:BX – адрес блока параметров EPB:

+00h (слово) – сегментный адрес окружения, которое будет скопировано для нового процесса (или 0, если используется текущее окружение);
+02h (4 байта) – адрес командной строки для нового процес- са;

+06h (4 байта) – адрес первого FCB для нового процесса; +0Ah (4 байта) – адрес второго FCB для нового процесса; +0Eh (4 байта) – здесь будет записан SS:SP нового процесса после его завершения (только для AL = 01);

+12h (4 байта) – здесь будет записан CS:IP (точка входа) но-

вого процесса после его завершения (только для AL = 01). AL = 03h – загрузить как оверлей:

DS:DX — адрес ASCIZ-строки с полным именем программы, ES:BX — адрес блока параметров:

+00h (слово) – сегментный адрес для загрузки оверлея,
+02h (слово) – число, которое будет использовано в командах, использующих непосредственные сегментные адреса (обычно то же самое число, что и в предыдущем поле или 0 для com-

файлов).
AL = 05h – подготовиться к выполнению (DOS 5.0+):

DS:DX – адрес следующей структуры: +00h(слово) – 00h,
+02h (слово) :

бит 0 – exe-программа,

бит 1 – программа-оверлей,
+04h (4 байта) – адрес ASCIZ-строки с именем новой про- граммы;
+08h (слово) – сегментный адрес PSP новой программы; +0Ah (4 байта) – точка входа новой программы;
+0Eh (4 байта) – размер программы, включая PSP.

Вывод: Если CF = 1, то произошла ошибка (в AX -- 02h – файл не найден,

-- 05h – доступ к файлу запрещен, -- 08h – не хватает памяти,
-- 0Ah – неправильное окружение, -- 0Bh – неправильный формат.

= код ошибки):

ЕслиCF = 0,тооперацияуспешновыполнена: BX и DX изменены.

Особенности: Для подфункций 00 и 01 требуется, чтобы было достаточно сво- бодной памяти для загрузки программы, поэтому com-программы должны воспользоваться функцией DOS 4Ah для уменьшения отведенного им блока памяти до минимально необходимого.

При вызове подфункции 03, DOS загружает оверлей в память, выделенную текущим процессом, так что exe-программы долж- ны убедиться, что ее достаточно.
Эта функция игнорирует расширение файла и различает exe- и com-файлы по первым двум байтам заголовка («MZ» для exe- файлов).

3. Оверлейные модули.

```
; адреса FCB программы
; длина командной строки
```

```
; командная строка (3)
; командная строки (122)
; длина программы
```

Оверлей – это часть исполняемой программы (обычно процедура, хотя это может быть полностью самостоятельная программа со своими сегментами данных и стека), которая по мере необходимости загружается в определенную область памяти. Различные оверлейные модули могут загружаться в одно и то же место, перекрывая предыдущий код, что позволяет экономить память, но снижает быстродействие программы при частых загрузках. Пример ассемблер- ного кода оверлейного модуля:

Код программы:

```
.MODEL tiny

.CODE

START:
   mov ax, @data
   mov ds, ax
   mov es, ax

   call PARSE_CMD


   xor cx, cx
   mov cl, TIMES_TO_RUN   ; count of runs programm

   cmp cl, 0
   je EXIT

   mov sp, PROGRAMM_LENGTH + 300h ; move stack na 200h
   mov ah, 4ah   ; set size of memory block
   STACK_SHIFT = PROGRAMM_LENGTH + 300h
   mov bx, STACK_SHIFT shr 5
   int 21h

   mov ax, cs
   mov word ptr EPB + 4,   ax      ; command string seg
   mov word ptr EPB + 8,   ax      ; first fcb seg
   mov word ptr EPB + 0ch, ax      ; second fcb seg

   lea dx, TIMES_TO_RUN_STR
   mov ah, 9
   int 21h

   RUN_PROGRAM_CYCLE:
      mov ax, 4b00h      ; load and run programm
      lea dx, PROGRAM_NAME  ; address of program
      lea bx, EPB   ; params EPB
      int 21h

      jc FATAL_ERROR

   loop RUN_PROGRAM_CYCLE

   int 20h

EXIT:
   mov ah, 4ch
   int 21h

FATAL_ERROR:
   mov ah, 9
   cmp al, 2
   je SET_FILE_NOT_FOUND
   cmp al, 5
```

```
    je SET_ACCESS_DENIED
    cmp al, 0bh
    je SET_NOT_VALID_FORMAT

SHOW_MESSAGE:
    int 21h
    jmp EXIT

SET_FILE_NOT_FOUND:
    lea dx, FILE_NOT_FOUND_MESSAGE
    jmp SHOW_MESSAGE

SET_ACCESS_DENIED:
    lea dx, ACCESS_DENIED_MESSAGE
    jmp SHOW_MESSAGE

SET_NOT_VALID_FORMAT:
    lea dx, NOT_VALID_FORMAT_MESSAGE
    jmp SHOW_MESSAGE

PARSE_CMD proc
    mov cl, [80h]
    cmp cl, 0
    je EMPTY_PARAMETER

    mov di, 82h

    call SKIP_SPACES
    lea si, PROGRAM_NAME
    call WRITE_PARAMETR

    call SKIP_SPACES
    lea si, TIMES_TO_RUN_STR
    call WRITE_PARAMETR
    call TO_DIGIT

    lea si, PARAMETER_FOR_RUN_PROGRAM
    call SKIP_SPACES

    dec di
    WRITE_PARAMETER_FOR_RUN_PROGRAM:
        cmp [di], 0dh
        je EXIT_FROM_PARSE

        mov al, [di]
        mov [si], al

        inc si
        inc di
    jmp WRITE_PARAMETER_FOR_RUN_PROGRAM

    EXIT_FROM_PARSE:
    ret

EMPTY_PARAMETER:
    mov ah, 9
    lea dx, EMPTY_PARAMETER_MESSAGE
    int 21h
    mov ah, 4ch
    int 21h
PARSE_CMD endp

WRITE_PARAMETR proc
     WRITE_CYCLE:
        mov al, [di]
        cmp al, 0
        je END_OF_WRITE
        cmp al, ' '
        je END_OF_WRITE
        cmp al, 9
        je END_OF_WRITE
        cmp al, 0dh
```

```
        je END_OF_WRITE
        mov [si], al
        inc di
        inc si
        jmp WRITE_CYCLE

    END_OF_WRITE:
    ret
WRITE_PARAMETR endp

SKIP_SPACES proc
    dec di
    CYCLE:
        inc di
        cmp [di], 0dh
        je END_OF_SKIPPING
        cmp [di], 0
        je END_OF_SKIPPING
        cmp [di], ' '
        je CYCLE
        cmp [di], 9
        je CYCLE
    END_OF_SKIPPING:
    ret
SKIP_SPACES endp

TO_DIGIT proc
    push di
    call IS_VALID_DIGIT

    lea di, TIMES_TO_RUN_STR
    xor bx, bx
    xor ax, ax
    mov bl, 10

    TO_DIGIT_CYCLE:
        mov cl, [di]
        sub cl, '0'

        mul bx
        add al, cl
        inc di
        cmp [di], '$'
    jne TO_DIGIT_CYCLE
    pop di
    mov TIMES_TO_RUN, al
    ret
TO_DIGIT endp

IS_VALID_DIGIT proc
    lea di, TIMES_TO_RUN_STR
    xor cx, cx
    mov cl, 4

    FIND_SIZE_CYCLE:
        cmp [di], '$'
        je EXIT_FROM_CYCLE
        cmp [di], '0'
        jl NOT_VALID_DIGIT
        cmp [di], '9'
        jg NOT_VALID_DIGIT

        inc di
    loop FIND_SIZE_CYCLE

    EXIT_FROM_CYCLE:

    lea di, TIMES_TO_RUN_STR
    cmp cl, 4
    je NOT_VALID_DIGIT

    cmp cl, 1
```

```
    jl NOT_VALID_DIGIT
    je IS_LOWER
    ret

IS_LOWER:
    cmp [di], '2'
    jg NOT_VALID_DIGIT
    cmp [di + 1], '5'
    jg NOT_VALID_DIGIT
    cmp [di + 2], '5'
    jg NOT_VALID_DIGIT
    ret

NOT_VALID_DIGIT:
    mov ah, 9
    lea dx, NOT_VALID_DIGIT_MESSAGE
    int 21h
    mov ah, 4ch
    int 21h

IS_VALID_DIGIT endp

LINE db 128 dup ('$')
TIMES_TO_RUN db 0
TIMES_TO_RUN_STR db 4 dup('$')
PROGRAM_NAME db 128 dup(0)
PARAMETER_FOR_RUN_PROGRAM 128 dup(0dh)

EPB dw 0000  ;WORD
    dw offset PARAMETER_FOR_RUN_PROGRAM ; command string address
    dw 005ch, 0, 006ch, 0     ; FCB adresses

NOT_VALID_DIGIT_MESSAGE db "Digit must be in [0, 255]$"
EMPTY_PARAMETER_MESSAGE db "You should enter name of program and quantity of starts$"
FILE_NOT_FOUND_MESSAGE db "Executable file not found$"
ACCESS_DENIED_MESSAGE db "Access denied$"
NOT_VALID_FORMAT_MESSAGE db "Not valid format message$"

PROGRAMM_LENGTH EQU $ - START
end START
```

Результат работы программы:

```
Z:\>mount c /users/petia/asm/laba77asik
Drive C is mounted as local directory /users/petia/asm/laba77asik/

Z:\>c:

C:\>noname.com laba1.exe
Digit must be in [0, 255]
C:\>noname.com laba1.exe 3
3hello!!!



hello!!!



hello!!!




C:\>
```