**МИНОБРНАУКИ РОССИИ**

**РГУ нефти и газа (НИУ) имени И.М. Губкина**

|  |  |
| --- | --- |
| Факультет | Комплексной безопасности ТЭК |
| Кафедра | Безопасности информационных технологий |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Оценка комиссии: |  | | | Рейтинг: |  |
| Подписи членов комиссии: | | | | | |
|  | |  |  | | |
| (подпись) | |  | (фамилия, имя, отчество) | | |
|  | |  |  | | |
| (подпись) | |  | (фамилия, имя, отчество) | | |
|  | | | | | |
| (дата) | | | | | |
|  | |  |  | | |

**КУРСОВАЯ РАБОТА**

|  |  |
| --- | --- |
| по дисциплине | Языки программирования низкого уровня |

|  |  |
| --- | --- |
| на тему | «Написание программы декодирования машинных кодов заданных команд в команды ассемблера: CMP,RCL,DAA» |

|  |  |  |  |
| --- | --- | --- | --- |
| «К ЗАЩИТЕ» |  | ВЫПОЛНИЛА |  |
|  |  | Студентка группы | **КФ-12-03** |
|  |  |  | (номер группы) |
|  |  | Пронина О.В. | |
| (должность, ученая степень; фамилия, и.о.) |  | (фамилия, имя, отчество) | |
|  |  |  | |
| (подпись) |  | (подпись) | |
|  |  |  | |
| (дата) |  | (дата) | |

|  |  |
| --- | --- |
|  | Москва, 2023 |

МИНОБРНАУКИ РОССИИ

РГУ нефти и газа (НИУ) имени И.М. Губкина

|  |  |
| --- | --- |
| Факультет | Комплексной безопасности ТЭК |
| Кафедра | Безопасности информационных технологий |

**ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ**

|  |  |
| --- | --- |
| по дисциплине | Языки программирования низкого уровня |

|  |  |
| --- | --- |
| на тему | «Написание программы декодирования машинных кодов заданных команд в команды ассемблера: CMP,RCL,DAA» |

|  |  |  |  |
| --- | --- | --- | --- |
| ДАНО студентке | **Прониной Ольге Васильевне** | группы | **КФ-21-03** |
|  | (фамилия, имя, отчество в дательном падеже) |  | (номер группы) |

**Рекомендуемая литература:**

|  |  |
| --- | --- |
| 1 | [242691-001\_Pentium\_Pro\_Family\_Developers\_Manual\_Volume\_2\_Jan96.pdf (gubkin.ru)](https://edu.gubkin.ru/pluginfile.php/1217229/mod_resource/content/0/242691-001_Pentium_Pro_Family_Developers_Manual_Volume_2_Jan96.pdf) |
| 2 | [edu.gubkin.ru/pluginfile.php/1217230/mod\_resource/content/0/242692-001\_Pentium\_Pro\_Family\_Developers\_Manual\_Volume\_3\_Jan96.pdf](https://edu.gubkin.ru/pluginfile.php/1217230/mod_resource/content/0/242692-001_Pentium_Pro_Family_Developers_Manual_Volume_3_Jan96.pdf) |

**Требования к представлению результатов:**

|  |  |
| --- | --- |
| ü | Электронная версия |
|  | Бумажный вариант и электронный образ документа |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Руководитель: |  |  |  |  |  |  |  |
|  | (уч.степень) |  | (должность) |  | (подпись) |  | (фамилия, имя, отчество) |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Задание принял к исполнению: | студент |  |  |  |  |
|  | |  | (подпись) |  | (фамилия, имя, отчество) |

**ОГЛАВЛЕНИЕ**

[1 Дизассемблирование 4](#__RefHeading___121)

[2 Структура машинной команды 5](#__RefHeading___142)

[2.1 Префиксы перед кодом операции 5](#__RefHeading___123)

[2.2 Префиксы команды 5](#__RefHeading___124)

[2.3 Префикс изменения размера адреса 5](#__RefHeading___125)

[2.4 Префикс изменения размера операнда 6](#__RefHeading___126)

[2.5 Префиксы замены сегмента 6](#__RefHeading___127)

[2.6 Код операции 6](#__RefHeading___128)

[2.7 Байт MRM - (mod,reg,r/m) 7](#__RefHeading___129)

[2.8 Байт SIB - (scale,index,base) 8](#__RefHeading___130)

[2.9 Непосредственный операнд 9](#__RefHeading___131)

[3 Декодирование команд CMP,RCL,DAA 10](#__RefHeading___132)

[3.1 DAA 10](#__RefHeading___133)

[3.2 RCL 10](#__RefHeading___134)

[3.3 CMP 11](#__RefHeading___135)

[ПРИЛОЖЕНИЕ 1 – ФАЙЛ «disasm» 12](#__RefHeading___136)

[ПРИЛОЖЕНИЕ 2 – ФАЙЛ «COMMANDS» 32](#__RefHeading___137)

# 1 Дизассемблирование

Процесс дизассемблирования – это процесс преобразования машинного кода, который представляет собой набор инструкций, написанных на языке машины, в исходный код на языке ассемблера.Дизассемблирование может быть выполнено с использованием специальных программных инструментов, называемых дизассемблерами. Дизассемблеры анализируют бинарный файл, содержащий машинный код, и пытаются преобразовать его в читаемый исходный код на языке ассемблера.

Процесс дизассемблирования включает в себя следующие шаги:

* чтение бинарного файла: дизассемблер начинает с чтения бинарного файла, содержащего машинный код программы. Этот файл может быть исполняемым файлом, библиотекой или другим типом файла, содержащим машинный код;
* анализ инструкций: дизассемблер анализирует каждую инструкцию в бинарном файле и пытается определить ее тип и параметры. Это включает в себя определение операций, регистров, адресов памяти и других элементов, используемых в инструкции;
* преобразование в ассемблерный код: на основе анализа инструкций, дизассемблер преобразует каждую инструкцию в соответствующий ассемблерный код. Ассемблерный код представляет собой текстовое представление инструкций на языке ассемблера, который более понятен для человека;
* восстановление структуры программы:дизассемблер также пытается восстановить структуру программы, определяя функции, подпрограммы и другие блоки кода. Это может включать в себя определение точек входа, вызовов функций и переходов между различными частями программы;
* в результате процесса дизассемблирования получается исходный код на языке ассемблера, который может быть прочитан и понят человеком.

# 2 Структура машинной команды

## 2.1 Префиксы перед кодом операции

Собственно команда начинается с кода операции. Но перед командой может быть префикс. И даже сразу несколько префиксов. Каждый префикс - это один байт.Когда перед кодом операции помещаются префиксы, должны соблюдаться следующие два правила:

* если в команде есть более одного префикса, то префиксы должны располагаться в том порядке, в котором они показаны в таблицах, приведенных выше;
* в одной команде не могут стоять сразу два префикса одной группы (сразу два префикса команды или сразу два префикса замены сегмента).

Замечено, что процессоры обычно бывают терпимы к нарушению этих правил. Однако, стоит все же считать, что результат таких нарушений может оказаться непредсказуемым.

Префикс действует только в пределах той команды, перед которой он стоит.

## 2.2 Префиксы команды

Код F0 - префикс блокировки шины, команда LOCK ("lock" - запирать). Этот префикс употребляется только с теми командами, которые поддерживают такую возможность - возможность блокировки шины.

Коды F2, F3 - префиксы повторения, команда REP ("repeat" - повторять) и другие команды этой группы. Такие префиксы употребляются только с цепочечными командами. Префиксы группы REP позволяют организовать циклическое выполнение цепочечной команды.

Код F1 - нет такого префикса и нет такой команды. Тут следует заметить, что это единственный код для первого байта команды, который все еще остается свободным.

## 2.3 Префикс изменения размера адреса

Код 67.Если общий режим выполнения программы равен "32 бита", то для команды, перед которой есть префикс 67, устанавливается атрибут размера адреса "16" бит.

Если общий режим выполнения программы равен "16 бит", то для команды, перед которой есть префикс 67, устанавливается атрибут размера адреса "32" бита.

Действие префикса зависит от конкретной команды.

## 2.4 Префикс изменения размера операнда

Код 66.Если общий режим выполнения программы равен "32 бита", то для команды, перед которой есть префикс 66, устанавливается атрибут размера операнда "16" бит.

Если общий режим выполнения программы равен "16 бит", то для команды, перед которой есть префикс 66, устанавливается атрибут размера операнда "32" бита. Действие префикса зависит от конкретной команды.

## 2.5 Префиксы замены сегмента

Для команд, в которых тем или иным способом задается адрес памяти, этот адрес памяти всегда относится к некоторому сегменту, заданному по умолчанию. С помощью префикса замены сегмента можно изменить сегмент, заданный по умолчанию. Однако, такая замена возможна не во всех случаях - это зависит от конкретной команды.

Код 26 - сегмент по умолчанию заменяется на сегмент ES.  
Код 2E - сегмент по умолчанию заменяется на сегмент CS.  
Код 36 - сегмент по умолчанию заменяется на сегмент SS.  
Код 3E - сегмент по умолчанию заменяется на сегмент DS.  
Код 64 - сегмент по умолчанию заменяется на сегмент FS.  
Код 65 - сегмент по умолчанию заменяется на сегмент GS.

## 2.6 Код операции

После всех префиксов (если перед командой есть префиксы) начинается собственно машинная команда. И начинается машинная команда с кода операции.

Поле с кодом операции всегда присутствует в любой машинной команде. Очевидно, что в команде должен быть хотя бы один байт. И тогда этим единственным байтом как раз будет именно код операции.

Код операции (сокращенно - КОП) может состоять из одного байта или из двух байт. Если первый байт кода операции имет значение 0F, то в этом коде операции есть еще и второй байт.

Если КОП состоит из одного байта, то этот байт и является основным байтом. Если КОП состоит из двух байт, то основным следует считать второй байт кода операции.

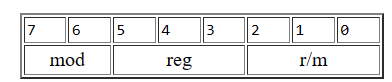
## 2.7 Байт MRM - (mod,reg,r/m)

Байт MRM, полное название ( mod,reg,r/m ) - это байт режима адресации.

Полное название выглядит именно так - "байт ( mod,reg,r/m )". А сокращенное название в разных книгах выглядит по-разному. Байт MRM имеется не во всех командах. Это определяется конкретным кодом операции (а говоря точнее, "основным кодом операции"), входит ли байт MRM в состав данной машинной команды или не входит.

Первым в этой группе идет байт ( mod,reg,r/m ), то есть, собственно байт MRM, затем, если требуется, может идти второй байт режима адресации, это байт SIB, а затем, если требуется, может быть еще и поле для задания адреса.

Байт MRM делится на три битовых поля: двухбитовое поле ( mod ), трехбитовое поле ( reg ), трехбитовое поле ( r/m ).



**Рис.1 Байт MRM**

Поле ( reg ) определяет первый операнд команды, операнд-приемник (destination). Поле ( r/m ) определяет второй операнд команды, операнд источник (source). Обычно бывает именно такое распределение ролей между ( reg ) и ( r/m ), который из операндов будет источником, а который приемником. Но бывает и наоборот, это зависит от конкретной команды.

Поле ( reg ) задает регистр общего назначения. Три бита - это восемь вариантов, восемь разных регистров.

Поле ( r/m ) задает либо регистр, либо память. Это поле действует совместно с полем ( mod ), вместе будет пять битов и получается 32 варианта, это дает 8 вариантов для задания регистров и 24 варианта для задания формы и режима адресации памяти.

В режиме "32 бита" после байта MRM может стоять еще и второй байт режима адресации - байт SIB, так что количество разных форм задания адреса памяти еще более возрастает.

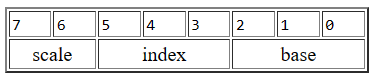
## 2.8 Байт SIB - (scale,index,base)

Байт SIB - это второй байт режима адресации.

Байт SIB возможен в команде только в режиме 32 бита и только в том случае, когда в команде уже есть байт MRM .

Этот дополнительный байт для задания режима адресации включается в состав команды только при некоторых определенных значениях в полях ( mod ) и ( r/m ) байта MRM.

Аналогично байту MRM, байт SIB делится на три битовых поля:



**Рис.2 Байт SIB**

Байт SIB позволяет применять еще более сложные формы задания адреса в памяти.

## 2.9 Поле для задания адреса

В этом поле из 1, или 2, или 4 байт, расположенном внутри команды, может быть задан либо полный адрес, либо смещение относительно некоторого адреса.

С точки зрения терминологии, не всегда можно отличить "смещение" от "полного адреса". Например, как называть полный адрес внутри сегмента, если такой адрес всегда задается смещением от начала сегмента.

Поле задания адреса может по-разному использоваться в разных командах. Однако, рассматривая общую структуру команды, важно различать следующие два случая:

Если в формат команды входит байт MRM, то поле для задания адреса (если оно есть) является приложением к байту MRM, оно может потребоваться в разных формах задания адреса, применяемых в байте MRM. Причем от кода байта MRM зависит, будет ли в данной машинной команде поле для задания адреса или не будет.

Если в формате команды нет байта MRM, то поле для задания адреса используется в той конкретной форме, которая предписана для данной конкретной машинной команды.

Если в формате команды есть обозначение MRM, то дополнительные байты, относящиеся к байту MRM никак не показываются. Так что поле для задания адреса в этом случае показано не будет.

## 2.10 Непосредственный операнд

Форматы многих команд предусматривают возможность задавать константу непосредственно внутри команды. На языке ассемблера это соответствует заданию для операнда конкретного численного значения вместо имени переменной.

Поле для непосредственного операнда есть не во всех машинных командах.

Если в формате команды есть поле для задания непосредственного операнда, то это поле в команде всегда будет последним.

# 3 Декодирование команд CMP,RCL,DAA

## 3.1 DAA

Команда с единственным opcode = 27h. Состоит из 1 байта. Не имеет операндов.Соответственно, файл читается побайтово. При попадении на байт 27h записывается DAA, и цикл переходит к следующему байту.

## 3.2 RCL

Имеет 2 операнда. Опкоды: D0, D2, C0, D1, D3, C1 - в зависимости от битности и второго операнда.

Если опкод - D0 или D1, то второй операнд – единица: rcl ax, 1

Если опкод - D2 / D3, второй операнд - регистр CL:rcl ax, cl

При оставшихся кодах C0 / C1, второй операнд - встроенное число:rcl ax, 7

При этом команда читается следующим образом:

0 1 2 --- байты

opcode ModR/M (imm8)

где imm8 - встроенное число. Если opcode не равен C0 и С1, то этого числа нет.

Байт ModR/M введён в документации Intel. Вкратце, побитово он выглядит так:

7 6 5 4 3 2 1 0

\\_\_\_/ \\_\_\_\_\_\_\_\_\_/ \\_\_\_\_\_\_\_\_\_/

Mod Reg/opcode R/M (register/memory)

Mod - определяет вид и размер операндов, можно сказать, подвид инструкции.

Reg/opcode может определять регистр или дополнять opcode (в RCL равен 2).

R/M определяет операнд регистра или взаимодействия с памятью.

Следующий код включает взаимодействие с памятью:

rcl word ptr [bx], 1

Такая инструкция кодируется следующим образом:

D1 17

Второй байт, ModR/M, раскладывается двоично как 00'010'111 (Mod == 00, R/M == 111).

То есть порядок чтения такой: сначала первый байт (всегда), затем байт ModR/M, затем - при необходимости - третий встроенный байт.

Декодирование идёт последовательно. В основном операнды определяются байтом ModR/M.

## 3.3 CMP

Имеет 2 операнда. Опкоды: 3C (I), 3D (I), 80 (MI), 83 (MI), 38 (MR), 39 (MR), 3A (RM), 3B (RM).

Порядок чтения:

1. opcode

2. ModR/M - при необходимости (определяется опкодом).

3. при необходимости (определяется опкодом) - встроенный байт или 2.

Из документации Intel: кодирование ModR/M байта

I - декодировка регистра не требуется (напр. `cmp ax, 2`). В этом случае регистр (первый операнд) всегда al/ax.

MI - требуется операнд памяти (напр. `cmp [bx\*2+1], 78`)

MR - Первый операнд декодируется по r/m части ModR/M, а второй - по части reg (0-2 и 3-5 биты соответственно)

RM - первый операнд декодируется по reg, а второй - по r/m части бита ModR/M.

Сводка:

3C: opcode, 1 байт

3D: opcode, 2 байта

80: opcode, ModR/M, 1 байт

83: opcode, ModR/M, 1 байт

38: opcode, ModR/M

39: opcode, ModR/M

3A: opcode, ModR/M

3B: opcode, ModR/M

В остальном алгоритм такой же, как при декодировании RCL.

# ПРИЛОЖЕНИЕ 1 – ФАЙЛ «disasm»

.model small

.stack 200h

.data

filename\_in db "input.bin",0 ; файл для чтения

filename\_out db "output.asm",0 ; файл для записи вывода

buffer dd 0 ; буфер для команд

o\_cmd db "$$$$" ; мнемоники всех команд, декодируемых этой программой, описываются 3 символами

m\_cmp db "cmp$"

m\_rcl db "rcl$"

m\_daa db "daa$"

m\_err db "ERR$"

o\_op1 db "$$$" ; первый операнд может быть только регистром

o\_op2 db "$$$$$$$$" ; здесь может быть регистр, число или обращение к памяти

o\_cmd\_full db " $" ; строка полной команды

; строки для расшифровки трёх бит первого операнда в ModR/M байте:

o\_op1decb db "alcldlblahchdhbh" ; при 1-байтовой вариации

o\_op1decw db "axcxdxbxspbpsidi" ; при 2-байтовой вариации

endl db 0dh,0ah,"$" ; символы перевода строки

errormsg db "Error$"

msg\_fopen\_err db "fopen error$"

msg\_fseek\_err db "fseek error$"

msg\_fread\_err db "fread error$"

msg\_fwrite\_err db "fwrite error$"

msg\_fcreate\_err db "fcreate error$"

; для удобного хранения файловых дескрипторов

inp\_filedesc dw 0

out\_filedesc dw 0

.code

mov ax, @data

mov ds, ax ; инициализация data segment

; 1. открыть файлы

lea dx, filename\_in ; адрес памяти filename\_in

mov al, 0 ; режим чтения

call fopen

mov word ptr inp\_filedesc, ax ; скопировать дескриптор файла

mov bx, ax

lea dx, filename\_out

mov cx, 0 ; нормальный файл (чтение/запись)

call fcreate

mov word ptr out\_filedesc, ax

; 2. узнать размер файла

call fsize

loop1:

test di, di

jnz loop1\_di\_notzero

test si, si

jz loop1\_end

dec si

loop1\_di\_notzero:

dec di

lea dx, buffer ; адрес памяти буфера

mov cx, 1 ; 1 байт для чтения

call fread

call disasm

lea ax, m\_err

cmp ax, dx

je loop1\_continue

lea dx, o\_cmd\_full

call printstr

call newline

call fwritestr

lea dx, endl ; fwritestr подходит также для записи перевода строки

call fwritestr

loop1\_continue:

jmp loop1

loop1\_end:

mov ah, 3eh ; служба DOS для закрытия файла

int 21h ; прерывание DOS

mov ah, 4ch ; завершить программу

int 21h

; записывает эффективный адрес в буфер

; работает при условии: Mod == 1, Mod == 2 или (Mod == 0 && cl != 6)

; аргументы:

; cl - представляет 3 бита R/M

; di - текущий указатель на буфер

; возвращаемые значения:

; di - указатель на конец записи в буфер

write\_eff\_addr proc

mov byte ptr [di], '['

inc di

test cl, 4 ; проверить, установлен ли 3-й бит в CL

jnz wea\_bit3

mov byte ptr [di], 'b'

inc di

test cl, 2 ; проверить 2-й бит

jnz wea\_bit2

mov byte ptr [di], 'x'

jmp wea\_1

wea\_bit2:

mov byte ptr [di], 'p'

jmp wea\_1

wea\_1:

inc di

mov byte ptr [di], '+'

inc di

; последний регистр теперь зависит только от 1 бита

test cl, 1

jnz wea\_2

mov byte ptr [di], 's'

jmp wea\_3

wea\_2:

mov byte ptr [di], 'd'

wea\_3:

inc di

mov byte ptr [di], 'i'

inc di

jmp wea\_end

wea\_bit3:

cmp cl, 5

je wea\_5

cmp cl, 6

je wea\_6

cmp cl, 7

je wea\_7

wea\_4:

; 2 байта в строке нужно менять местами из-за порядка байт LE

mov word ptr [di], 'is'

jmp wea\_pre

wea\_5:

mov word ptr [di], 'id'

jmp wea\_pre

wea\_6:

mov word ptr [di], 'pb'

jmp wea\_pre

wea\_7:

mov word ptr [di], 'xb'

jmp wea\_pre

wea\_pre:

add di, 2

wea\_end:

mov byte ptr [di], ']'

inc di

ret

write\_eff\_addr endp

; взятие команды из буфера и преобразование в мнемонику

; (учитывая операнды)

disasm proc

push si

push di

push bx

push cx

mov cl, byte ptr buffer

mov word ptr inp\_filedesc, bx

; сначала сравниваем первый байт с опкодами команд

cmp cl, 27h

je is\_daa

lea di, o\_op1decb ; если cmp сравнивает 1 байт, то первый операнд декодируется 1-байтовым регистром

cmp cl, 3ah

je is\_cmp

lea di, o\_op1decw ; иначе - 2-байтовым регистром

cmp cl, 3bh

je is\_cmp

; позже понадобится узнать подтип RCL по опкоду (если инструкция - RCL)

; удобно сохранить его в si

mov si, cx

; заранее устанавливаем указатель di на нужный массив

lea di, o\_op1decb

cmp cl, 0d0h

je is\_rcl\_near

cmp cl, 0d2h

je is\_rcl\_near

cmp cl, 0c0h

je is\_rcl\_near

lea di, o\_op1decw

cmp cl, 0d1h

je is\_rcl\_near

cmp cl, 0d3h

je is\_rcl\_near

cmp cl, 0c1h

je is\_rcl\_near

lea dx, m\_err

xor si, si

jmp disasm\_end

is\_daa:

lea dx, m\_daa

; при переходе на disasm\_end проверяется si

; si должен быть ненулевым, если у инструкции есть операнды, иначе 0

xor si, si

jmp disasm\_end

is\_rcl\_near:

jmp is\_rcl

is\_cmp:

; операнды декодируются, исходя из следующего байта - ModR/M.

; первые три бита обозначают второй операнд.

; четвертый-шестой биты - первый операнд.

; в зависимости от opcode операнды могут быть 1-байтными регистрами или 2-байтными.

; так, при cl == 3Ah, инструкция сравнивает 1 байт, а при cl == 3Bh - 2.

; также, при определенном значении ModR/M, следующим операндом может быть обращение к памяти.

; прочитать байт ModR/M:

lea dx, buffer+1

mov cx, 1

call fread

mov bx, dx

mov bl, [bx]

; первый операнд

mov cl, bl

rcr cl, 3 ; сдвиг на 3 байта вправо (нужные биты окажутся вначале)

and cx, 7 ; установка всех остальных бит в 0

push bx ; целый ModR/M пока не нужен

rcl cl, 1 ; умножение на 2

mov bx, di

add bx, cx

mov cx, [bx] ; в cx окажется декодированный первый операнд

; записать первый операнд в o\_op1

lea bx, o\_op1

mov [bx], cx

pop bx ; восстановить ModR/M в bl

; для второго операнда понадобится также Mod - 2 бита.

; сначала получим Mod:

mov ch, bl

rcr ch, 6

and ch, 3

; затем 3 бита R/M (register/memory operand):

mov cl, bl

and cl, 7

cmp ch, 1

je disasm\_cmp\_mod1

cmp ch, 2

je disasm\_cmp\_mod2

cmp ch, 3

je disasm\_cmp\_mod3

disasm\_cmp\_mod0:

lea di, o\_cmd\_full+8 ; установить указатель в di на начало 2-го операнда

cmp cl, 6

je disasm\_cmp\_mod0\_1 ; если cl == 6, то 2-й операнд декодируется по-другому

call write\_eff\_addr ; декодировать полностью второй операнд

jmp disasm\_cmp\_end

disasm\_cmp\_mod0\_1:

mov byte ptr [di], '['

inc di

; прочитать еще 2 байта

lea dx, buffer+2

mov cx, 2

call fread

mov bx, dx

mov cx, [bx]

; записать число после скобки

call writeword

mov byte ptr [di], 'h' ; метка шестнадцатеричного числа

inc di

; закрыть скобку (указатель на месте сразу после числа)

mov byte ptr [di], ']'

inc di

jmp disasm\_cmp\_end

disasm\_cmp\_mod1:

lea di, o\_cmd\_full+8

call write\_eff\_addr

jmp disasm\_cmp\_write\_disp8

disasm\_cmp\_mod2:

lea di, o\_cmd\_full+8

call write\_eff\_addr

jmp disasm\_cmp\_write\_disp16

disasm\_cmp\_mod3:

mov ch, 0 ; Mod больше не понадобится

rcl cl, 1 ; умножение на 2

mov bx, di

add bx, cx

mov cx, [bx] ; в cx окажется декодированный второй операнд

; записать полученный операнд после запятой

lea di, o\_cmd\_full+8

mov [di], cx

add di, 2

jmp disasm\_cmp\_end

disasm\_cmp\_write\_disp8:

; прочитать 1 байт

lea dx, buffer+2

mov cx, 1

call fread

mov bx, dx

mov cl, [bx]

; записать +

mov byte ptr [di], '+'

inc di

; записать число

call writebyte

mov byte ptr [di], 'h'

inc di

jmp disasm\_cmp\_end

disasm\_cmp\_write\_disp16:

; прочитать 2 байта

lea dx, buffer+2

mov cx, 2

call fread

mov bx, dx

mov cx, [bx]

; записать +

mov byte ptr [di], '+'

inc di

; записать число

call writeword

mov byte ptr [di], 'h'

inc di

disasm\_cmp\_end:

mov byte ptr [di], '$'

lea dx, m\_cmp

lea di, o\_cmd\_full+3

mov byte ptr [di], ' '

inc di

lea si, o\_op1

mov si, [si]

mov [di], si

add di, 2

mov byte ptr [di], ',' ; запятая после первого операнда

mov si, 1

jmp disasm\_end

is\_rcl:

; прочитать байт ModR/M:

lea dx, buffer+1

mov cx, 1

call fread

mov bx, dx

mov bl, [bx]

; выделить 3 бита R/M:

mov cl, bl

and cl, 7

; получить Mod:

mov ch, bl

rcr ch, 6

and ch, 3

cmp ch, 3

; во всех случаях, кроме Mod == 3, нужно вывести тип указателя

je disasm\_rcl\_mod3\_near

jmp disasm\_rcl\_not\_mod3

disasm\_rcl\_mod3\_near:

jmp disasm\_rcl\_mod3

disasm\_rcl\_not\_mod3:

lea di, o\_cmd\_full+4 ; установить указатель в di на начало 1-го операнда

; выделить опкод из предварительно сохраненного si

mov dx, si

cmp dl, 0d0h

je is\_rcl\_byte\_variation

cmp dl, 0d2h

je is\_rcl\_byte\_variation

cmp dl, 0c0h

je is\_rcl\_byte\_variation

is\_rcl\_word\_variation:

; копирование строки по 2 байта - максимально эффективное для

; 16-битных процессоров

mov word ptr [di], 'ow'

add di, 2

mov word ptr [di], 'dr'

add di, 2

jmp disasm\_rcl\_1

is\_rcl\_byte\_variation:

mov word ptr [di], 'yb'

add di, 2

mov word ptr [di], 'et'

add di, 2

disasm\_rcl\_1:

mov word ptr [di], 'p '

add di, 2

mov word ptr [di], 'rt'

add di, 2

mov word ptr [di], ' '

inc di

cmp ch, 1

je disasm\_rcl\_mod1

cmp ch, 2

je disasm\_rcl\_mod2

disasm\_rcl\_mod0:

cmp cl, 6

je disasm\_rcl\_mod0\_1 ; если cl == 6, то 2-й операнд декодируется по-другому

call write\_eff\_addr ; декодировать полностью 1-й операнд

jmp disasm\_rcl\_end

disasm\_rcl\_mod0\_1:

mov byte ptr [di], '['

inc di

; прочитать еще 2 байта

lea dx, buffer+2

mov cx, 2

call fread

mov bx, dx

mov cx, [bx]

; записать число после скобки

call writeword

mov byte ptr [di], 'h' ; метка шестнадцатеричного числа

inc di

; закрыть скобку (указатель на месте сразу после числа)

mov byte ptr [di], ']'

inc di

jmp disasm\_rcl\_end

disasm\_rcl\_mod1:

call write\_eff\_addr

jmp disasm\_rcl\_write\_disp8

disasm\_rcl\_mod2:

call write\_eff\_addr

jmp disasm\_rcl\_write\_disp16

disasm\_rcl\_mod3:

mov ch, 0

rcl cl, 1 ; умножение на 2

mov bx, di

add bx, cx

mov cx, [bx] ; в cx окажется декодированный первый операнд

; записать полученный операнд

lea di, o\_cmd\_full+4

mov [di], cx

add di, 2

jmp disasm\_rcl\_end

disasm\_rcl\_write\_disp8:

; прочитать 1 байт

lea dx, buffer+2

mov cx, 1

call fread

mov bx, dx

mov cl, [bx]

; записать +

mov byte ptr [di], '+'

inc di

; записать число

call writebyte

mov byte ptr [di], 'h'

inc di

jmp disasm\_rcl\_end

disasm\_rcl\_write\_disp16:

; прочитать 2 байта

lea dx, buffer+2

mov cx, 2

call fread

mov bx, dx

mov cx, [bx]

; записать +

mov byte ptr [di], '+'

inc di

; записать число

call writeword

mov byte ptr [di], 'h'

inc di

disasm\_rcl\_end:

; записать запятую с пробелом

mov byte ptr [di], ','

inc di

mov byte ptr [di], ' '

inc di

; выделить опкод из предварительно сохраненного si

mov dx, si

cmp dl, 0d0h

je is\_rcl\_1\_variation

cmp dl, 0d1h

je is\_rcl\_1\_variation

cmp dl, 0d2h

je is\_rcl\_cl\_variation

cmp dl, 0d3h

je is\_rcl\_cl\_variation

is\_rcl\_imm8\_variation: ; подтип RCL,

; использующий один дополнительный байт

; в качестве операнда (immediate 8-bit value)

; прочитать 1 байт

lea dx, buffer+2

mov cx, 1

call fread

mov bx, dx

mov cl, [bx]

; записать число

call writebyte

mov byte ptr [di], 'h'

inc di

jmp disasm\_rcl\_end1

is\_rcl\_1\_variation: ; подтип, сдвигающий только на 1 бит

mov byte ptr [di], '1'

inc di

jmp disasm\_rcl\_end1

is\_rcl\_cl\_variation: ; подтип, использующий регистр CL в качестве операнда

mov word ptr [di], 'lc'

add di, 2

disasm\_rcl\_end1:

mov byte ptr [di], '$'

lea dx, m\_rcl

mov si, 1

disasm\_end:

; переписать мнемонику в строку команды

lea di, o\_cmd\_full

mov bx, dx

mov cx, [bx]

mov [di], cx

add di, 2

add bx, 2

mov cl, [bx]

mov [di], cl

cmp si, 0

jne disasm\_end1

inc di

mov byte ptr [di], '$'

disasm\_end1:

pop cx

pop bx

pop di

pop si

ret

disasm endp

; вывод 1 байта в шестнадцатеричном формате

; аргументы:

; cl - байт, который будет напечатан

printbyte proc

; ax, dx изменяются в ходе процедуры

push ax

push dx

mov ah, 2 ; служба вывода символа

mov dl, cl

shr dl, 4

cmp dl, 9

jle prb1

add dl, 'A'-10

jmp prb2

prb1:

add dl, '0'

prb2:

int 21h

mov dl, cl

and dl, 15

cmp dl, 9

jle prb3

add dl, 'A'-10

jmp prb4

prb3:

add dl, '0'

prb4:

int 21h

; восстанавливаем значения регистров, как до вызова процедуры

pop dx

pop ax

ret

printbyte endp

; запись 1 байта в память в шестнадцатеричном формате

; аргументы:

; cl - байт, который будет напечатан

; di - указатель в начало записи

writebyte proc

push dx

mov dl, cl

shr dl, 4

cmp dl, 9

jle wb1

add dl, 'A'-10

jmp wb2

wb1:

add dl, '0'

wb2:

mov byte ptr [di], dl

inc di

mov dl, cl

and dl, 15

cmp dl, 9

jle wb3

add dl, 'A'-10

jmp wb4

wb3:

add dl, '0'

wb4:

mov byte ptr [di], dl

inc di

pop dx

ret

writebyte endp

; вывод 2-байтного числа в шестнадцатеричном формате

; аргументы:

; ax - число для вывода

printword proc

push cx

mov cl, ah

call printbyte

mov cl, al

call printbyte

pop cx

ret

printword endp

; запись 2-байтного числа в память в шестнадцатеричном формате

; аргументы:

; cx - число для вывода

; di - указатель в начало записи

writeword proc

push cx

mov cl, ch

call writebyte

pop cx

call writebyte

ret

writeword endp

; напечатать строку по адресу dx

printstr proc

push ax

mov ah, 9 ; служба вывода строки

int 21h

pop ax

ret

printstr endp

; выполнить переход на новую строку

newline proc

push ax

push dx

mov ah, 9

lea dx, endl

int 21h

pop dx

pop ax

ret

newline endp

; открытие файла

; параметры:

; dx - указатель на строку с названием файла

; al - режим доступа

; возвращаемые значения:

; ax - файловый дескриптор

fopen proc

mov ah, 3dh ; служба открытия файла

int 21h ; прерывание DOS

; при возникновении ошибки будет установлен carry flag

jc fopen\_err

ret

fopen\_err:

lea dx, msg\_fopen\_err

call printstr

call newline

jmp error\_occured

fopen endp

; создание файла

; параметры:

; dx - указатель на строку с названием файла

; cx - атрибуты

; возвращаемые значения:

; ax - файловый дескриптор

fcreate proc

mov ah, 3ch ; служба создания файла

int 21h ; прерывание DOS

jc fopen\_err

ret

fcreate\_err:

lea dx, msg\_fcreate\_err

call printstr

call newline

jmp error\_occured

fcreate endp

; чтение из файла

; параметры:

; bx - файловый дескриптор

; cx - количество байт для чтения

; dx - адрес буфера для прочитанных данных

; возвращаемые значения:

; ax - количество прочитанных байт

fread proc

push bx

mov bx, word ptr inp\_filedesc

mov ah, 3fh ; служба чтения файла

int 21h

pop bx

jc fread\_err

ret ; завершение функции, если ошибки нет

fread\_err:

lea dx, msg\_fread\_err

call printstr

call newline

jmp error\_occured

fread endp

; перемещение указателя файла

; параметры:

; bx - файловый дескриптор

; cx:dx - смещение в байтах (32-битное знаковое целое число)

; al - точка отсчета:

; 0 - смещение от начала

; 1 - от текущего расположения

; 2 - с конца

; возвращаемые значения:

; dx:ax - новый указатель файла

fseek proc

push bx

mov bx, word ptr inp\_filedesc

mov ah, 42h

int 21h

pop bx

jc fseek\_err

ret

fseek\_err:

lea dx, msg\_fseek\_err

call printstr

call newline

jmp error\_occured

fseek endp

; узнать размер файла

; параметры:

; bx - файловый дескриптор

; возвращаемые значения:

; si:di - размер файла

fsize proc

; сохранить регистры

push ax

push cx

push dx

; 1. получить текущий указатель

xor cx, cx

xor dx, dx

mov al, 1

call fseek

; сохранить dx:ax - значение указателя

push dx

push ax

; 2. перейти в конец файла

xor cx, cx

xor dx, dx

mov al, 2

call fseek

; сохранить dx:ax - размер файла

mov si, dx

mov di, ax

; 3. восстановить исходное значение указателя

pop dx

pop cx

mov al, 0

call fseek

; восстановить регистры

pop dx

pop cx

pop ax

ret

fsize endp

; записать в файл строку по адресу dx

; размер строки определяется расположением символа $

; аргументы:

; dx - адрес буфера

; возвращаемые значения:

; ax - количество записанных байт

fwritestr proc

push cx

push bx

mov bx, dx

xor cx, cx ; счетчик циклов

fwritestr\_loop:

mov al, [bx] ; получить символ

cmp al, '$' ; сравнить с $

je fwritestr\_loop\_end

; инкрементировать счетчик и указатель

inc bx

inc cx

jmp fwritestr\_loop

fwritestr\_loop\_end:

mov bx, word ptr out\_filedesc

mov ah, 40h

int 21h

pop bx

pop cx

jc error\_occured

ret

fwritestr endp

error\_occured:

; напечатать сообщение об ошибке и выйти

mov ah, 9 ; служба печати строки

lea dx, errormsg

int 21h

mov ah, 4ch ; служба завершения программы

int 21h

end

# ПРИЛОЖЕНИЕ 2 – ФАЙЛ «COMMANDS»

.model small

.stack 200h

.data

notbuffer dw 0,1,2,3,4,5,6,7,8

buffer dw 0

.code

cmp ax, ax

cmp ax, bx

cmp ax, cx

cmp ax, dx

cmp bx, ax

cmp bx, bx

cmp bx, cx

cmp bx, dx

cmp cx, ax

cmp cx, bx

cmp cx, cx

cmp cx, dx

cmp dx, ax

cmp dx, bx

cmp dx, cx

cmp dx, dx

cmp al, al

cmp al, bl

cmp al, cl

cmp al, dl

cmp bl, al

cmp bl, bl

cmp bl, cl

cmp bl, dl

cmp cl, al

cmp cl, bl

cmp cl, cl

cmp cl, dl

cmp dl, al

cmp dl, bl

cmp dl, cl

cmp dl, dl

cmp ah, ah

cmp ah, bh

cmp ah, ch

cmp ah, dh

cmp bh, ah

cmp bh, bh

cmp bh, ch

cmp bh, dh

cmp ch, ah

cmp ch, bh

cmp ch, ch

cmp ch, dh

cmp dh, ah

cmp dh, bh

cmp dh, ch

cmp dh, dh

cmp ax, si

cmp ax, di

cmp ax, sp

cmp ax, bp

cmp bx, si

cmp bx, di

cmp bx, sp

cmp bx, bp

cmp cx, si

cmp cx, di

cmp cx, sp

cmp cx, bp

cmp dx, si

cmp dx, di

cmp dx, sp

cmp dx, bp

cmp ax, word ptr buffer

cmp bx, word ptr buffer

cmp cx, word ptr buffer

cmp dx, word ptr buffer

cmp si, word ptr buffer

cmp di, word ptr buffer

cmp sp, word ptr buffer

cmp bp, word ptr buffer

cmp al, byte ptr buffer

cmp bl, byte ptr buffer

cmp cl, byte ptr buffer

cmp dl, byte ptr buffer

cmp al, [bx+si]

cmp al, [bx+si]+3

cmp ax, ax

rcl ax, cl

rcl bx, cl

rcl cx, cl

rcl dx, cl

rcl ax, 1

rcl bx, 1

rcl cx, 1

rcl dx, 1

rcl byte ptr [bx], 1

rcl word ptr buffer, 2

rcl word ptr [bx+si]+03, 1

rcl word ptr [bx+si]+0000h, cl

daa

end