Отчет по лабораторной работе №7

Дисциплина архитектура компьютера

Ахатов Эмиль Эрнстович

Содержание

Список иллюстраций

1 Цель работы

Изучение команд условного и безусловного переходов. Приобретение навыков написания программ с использованием переходов. Знакомство с назначением и структурой файла листинга.

2 Задание

1. Программа с использованием инструкции jmp(листинг 1)
2. Программа с использованием инструкции jmp(листинг 2)
3. Изучение структуры файла листинга
4. Выполнение заданий для самостоятельной работы

3 Теоретическое введение

Для реализации ветвлений в ассемблере используются так называемые команды передачи управления или команды перехода. Можно выделить 2 типа переходов: • условный переход – выполнение или не выполнение перехода в определен- ную точку программы в зависимости от проверки условия. • безусловный переход – выполнение передачи управления в определенную точку программы без каких-либо условий.

Безусловный переход выполняется инструкцией jmp (от англ. jump – прыжок), которая включает в себя адрес перехода, куда следует передать управление: mp Адрес перехода может быть либо меткой, либо адресом области памяти, в которую предварительно помещен указатель перехода. Кроме того, в качестве операнда можно использовать имя регистра, в таком случае переход будет осуществляться по адресу, хранящемуся в этом регистре.

Тип операнда	Описание
`jmp label`	Переход на метку labe
`mp [label]`	Переход по адресу в памяти, помеченному меткой label
`jmp eax`	Переход по адресу из регистра eax

В следующем примере рассмотрим использование инструкции jmp:

```
label:
    ...         ;
    ...         ; команды
    ...         ;
    jmp  label
```

Рис. 3.1: Использование инструкции jmp

Команды условного перехода__

Как отмечалось выше, для условного перехода необходима проверка какого-либо условия. В ассемблере команды условного перехода вычисляют условие перехода анализируя флаги из регистра флагов.

Регистр флагов

Флаг – это бит, принимающий значение 1 («флаг установлен»), если выполнено некоторое условие, и значение 0 («флаг сброшен») в противном случае. Флаги работают независимо друг от друга, и лишь для удобства они помещены в единый регистр — регистр флагов, отражающий текущее состояние процессора. В следующей таблице указано положение битовых флагов в регистре флагов:

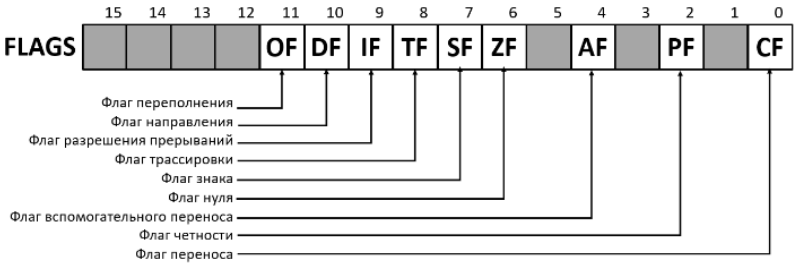


Рис. 3.2: Регистр флагов

Флаги состояния (биты 0, 2, 4, 6, 7 и 11) отражают результат выполнения ариф-

метических инструкций, таких как ADD, SUB, MUL, DIV.

Бит	Обозначение	Название	Описание
0	CF	Carry Flag - Флаг переноса	Устанавливается в 1, если при выполнении предыдущей операции произошёл перенос из старшего бита или если требуется заём (при вычитании). Иначе установлен в 0.
2	PF	Parity Flag - Флаг чётности	Устанавливается в 1, если младший байт результата предыдущей операции содержит чётное количество битов, равных 1.
4	AF	Auxiliary Carry Flag - Вспомогательный флаг переноса	Устанавливается в 1, если в результате предыдущей операции произошёл перенос (или заём) из третьего бита в четвёртый.
6	ZF	Zero Flag - Флаг нуля	Устанавливается 1, если результат предыдущей команды равен 0.
7	SF	Sign Flag - Флаг знака	Равен значению старшего значащего бита результата, который является знаковым битом в знаковой арифметике.
11	SF	Overflow Flag - Флаг переполнения	Устанавливается в 1, если целочисленный результат слишком длинный для размещения в целевом операнде (регистре или ячейке памяти).

Рис. 3.3: Регистр флагов

Описание инструкции cmp

Инструкция cmp является одной из инструкций, которая позволяет сравнить операнды и выставляет флаги в зависимости от результата сравнения. Инструкция cmp является командой сравнения двух операндов и имеет такой же формат, как и команда вычитания: cmp , Команда cmp, так же как и команда вычитания, выполняет вычитание - , но результат вычитания никуда не записывается и единственным результатом команды сравнения является формирование флагов. Примеры:

```
cmp ax,'4'      ; сравнение регистра ax с символом 4
cmp ax,4        ; сравнение регистра ax со значением 4
cmp al,cl       ; сравнение регистров al и cl
cmp [buf],ax    ; сравнение переменной buf с регистром ax
```

Рис. 3.4: Описание инструкции cmp

Описание команд условного перехода

Команда условного перехода имеет вид j label Мнемоника перехода связана со

значением анализируемых флагов или со способом формирования этих флагов. В табл. 8.3. представлены команды условного перехода, которые обычно ставятся после команды сравнения cmp. В их мнемокодах указывается тот результат сравнения, при котором надо делать переход. Мнемоники, идентичные по своему действию, написаны в таблице через дробь (например, ja и jnbe). Программист выбирает, какую из них применить, чтобы получить более простой для понимания текст программы.

Инструкции условной передачи управления по результатам арифметического сравнения cmp a,b

Типы операндов	Мнемокод	Критенрий условного перехода $a \vee b$	Значения флагов	Комментарий
Любые	`JE`	$a = b$	ZF = 1	Переход если равно
Любые	`JNE`	$a \neq b$	ZF = 0	Переход если не равно
Со знаком	`JL/JNGE`	$a < b$	SF $\neq$ OF	Переход если меньше
Со знаком	`JLE/JNG`	$a \leq b$	SF $\neq$ OF или ZF = 1	Переход если меньше или равно
Со знаком	`JG/JNLE`	$a > b$	SF = OF и ZF = 0	Переход если больше
Со знаком	`JGE/JNL`	$a \geq b$	SF = OF	Переход если больше или равно
Без знака	`JB/JNAE`	$a < b$	CF = 1	Переход если ниже

Рис. 3.5: Инструкции условной передачи управления по результатам арифметического сравнения cmp a,b

Типы операндов	Мнемокод	Критенрий условного перехода $a \vee b$	Значения флагов	Коммента-рий
Без знака	JBE/JNA	$a \leq b$	CF = 1 или ZF = 1	Переход если ниже или равно
Без знака	JA/JNBE	$a > b$	CF = 0 и ZF = 0	Переход если выше
Без знака	JAE/JNB	$a \geq b$	CF = 0	Переход если выше или равно

Рис. 3.6: Инструкции условной передачи управления по результатам арифметического сравнения cmp a,b

Примечание: термины «выше» («a» от англ. «above») и «ниже» («b» от англ. «below») применимы для сравнения беззнаковых величин (адресов), а термины «больше» («g» от англ. «greater») и «меньше» («l» от англ. «lower») используются при учёте знака числа. Таким образом, мнемонику инструкции JA/JNBE можно расшифровать как «jump if above (переход если выше) / jump if not below equal (переход если не меньше или равно)». Помимо перечисленных команд условного перехода существуют те, которые которые можно использовать после любых команд, меняющих значения флагов.

Мнемо-код	Значение флага для осуществления перехода	Мнемо-код	Значение флага для осуществления перехода
JZ	ZF = 1	JNZ	ZF = 0
JS	SF = 1	JNS	SF = 0
JC	CF = 1	JNC	CF = 0
JO	OF = 1	JNO	OF = 0
JP	PF = 1	JNP	PF = 0

Рис. 3.7: Инструкции условной передачи управления

В качестве примера рассмотрим фрагмент программы, которая выполняет умножение переменных a и b и если произведение превосходит размер байта, передает управление на метку Error. mov al, a mov bl, b mul bl jc Error

Файл листинга и его структура

Листинг (в рамках понятийного аппарата NASM) — это один из выходных файлов, создаваемых транслятором. Он имеет текстовый вид и нужен при отладке программы, так как кроме строк самой программы он содержит дополнительную информацию. Ниже приведён фрагмент файла листинга.

```
10 00000000 B804000000          mov eax,4
11 00000005 BB01000000          mov ebx,1
12 0000000A B9[00000000]        mov ecx,hello
13 0000000F BA0D000000          mov edx,helloLen
14
15 00000014 CD80                int 80h
```

Рис. 3.8: Фрагмент файла листинга

Строки в первой части листинга имеют следующую структуру

Рис. 3.9: Структура листинга

Все ошибки и предупреждения, обнаруженные при ассемблировании, транслятор выводит на экран, и файл листинга не создаётся. Итак, структура листинга: • номер строки — это номер строки файла листинга (нужно помнить, что номер строки в файле листинга может не соответствовать номеру строки в файле с исходным текстом программы); • адрес — это смещение машинного кода от начала текущего сегмента; • машинный код представляет собой ассемблированную исходную строку в виде шестна- дцатеричной последовательности. (например, инструкция int 80h начинается по сме- щению 00000020 в сегменте кода; далее идёт машинный код, в который ассемблируется инструкция, то есть инструкция int 80h ассемблируется в CD80 (в шестнадцатеричном представлении); CD80 — это инструкция на машинном языке, вызывающая прерывание ядра);

4 Выполнение лабораторной работы

4.1 Реализация переходов в NASM

создал каталог lab07 для программам лабораторной работы, перешёл в него и создал файл lab7-1.asm

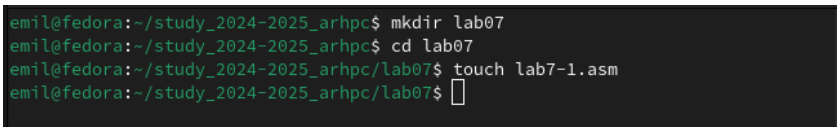


Рис. 4.1: Создание файла

4.2 Программа с использованием инструкции jmp

я скопировал внешний файл в созданный ката- лог, ввёл текст программы с использованием инструкции jmp в текстовый файл lab7-1.asm, создал объектный файл и проверил работу программы

```
%include 'in_out.asm' ; подключение внешнего файла
SECTION .data
msg1: DB 'Сообщение № 1',0
msg2: DB 'Сообщение № 2',0
msg3: DB 'Сообщение № 3',0
SECTION .text
GLOBAL _start
 _start:

jmp _label2

_label1:
  mov eax, msg1 ; Вывод на экран строки
  call sprintLF ; 'Сообщение № 1'
_label2:
  mov eax, msg2 ; Вывод на экран строки
  call sprintLF ; 'Сообщение № 2'
_label3:
   mov eax, msg3 ; Вывод на экран строки
   call sprintLF ; 'Сообщение № 3'
_end:
   call quit ; вызов подпрограммы завершения
```

Рис. 4.2: Редактирование файла

```
emil@fedora:~/study_2024-2025_arhpc/lab07$ nasm -f elf lab7-1.asm
emil@fedora:~/study_2024-2025_arhpc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
emil@fedora:~/study_2024-2025_arhpc/lab07$ ./lab7-1
Сообщение № 2
Сообщение № 3
```

Рис. 4.3: Запуск исполняемого файла

изменил текст программы и проверил её работу

```
%include 'in_out.asm' ; подключение внешнего файла
SECTION .data
msg1: DB 'Сообщение № 1',0
msg2: DB 'Сообщение № 2',0
msg3: DB 'Сообщение № 3',0
SECTION .text
GLOBAL _start
 _start:
jmp _label3
_label1:
  mov eax, msg1 ; Вывод на экран строки
  call sprintLF ; 'Сообщение № 1'
  jmp _end

_label2:
  mov eax, msg2 ; Вывод на экран строки
  call sprintLF ; 'Сообщение № 2'
  jmp _label1
_label3:
  mov eax, msg3 ; Вывод на экран строки
  call sprintLF ; 'Сообщение № 3'
  jmp _label2
_end:
  call quit ; вызов подпрограммы завершения
```

Рис. 4.4: Редактирование файла

```
emil@fedora:~/study_2024-2025_arhpc/lab07$ nasm -f elf lab7-1.asm
emil@fedora:~/study_2024-2025_arhpc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
emil@fedora:~/study_2024-2025_arhpc/lab07$ ./lab7-1
Сообщение № 3
Сообщение № 2
Сообщение № 1
```

Рис. 4.5: Запуск исполняемого файла

Программа, которая определяет и выводит на экран наибольшую из 3 целочисленных переменных: A,B и C.

Создаю файл с названием lab7-2.asm и ввожу текст программы.

```
%include 'in_out.asm'
section .data
msg1 db 'Введите B: ',0h
msg2 db "Наибольшее число: ",0h
A dd '20'
C dd '50'
section .bss
max resb 10
B resb 10
section .text
global _start
_start:
; ---------- Вывод сообщения 'Введите B: '
mov eax,msg1
call sprint
; ---------- Ввод 'B'
mov ecx,B
mov edx,10
call sread
; ---------- Преобразование 'B' из символа в число
mov eax,B
call atoi ; Вызов подпрограммы перевода символа в число
mov [B],eax ; запись преобразованного числа в 'B'
; ---------- Записываем 'A' в переменную 'max'
```

Рис. 4.6: Редактирование файла

При введении числа до 50, программа выводит наибольшее число 50, при введении числа больше 50, программа выводит введенное нами число. Программа сравнивает число A (значение 20) и C (значение 50) и инициализирует переменную max значением большего из них. Сравнивает текущее значение max с введённым числом B и обновляет max, если B больше. Выводит сообщение “Наибольшее число:” и затем значение переменной max, которая содержит наибольшее из трёх чисел: A, B и C

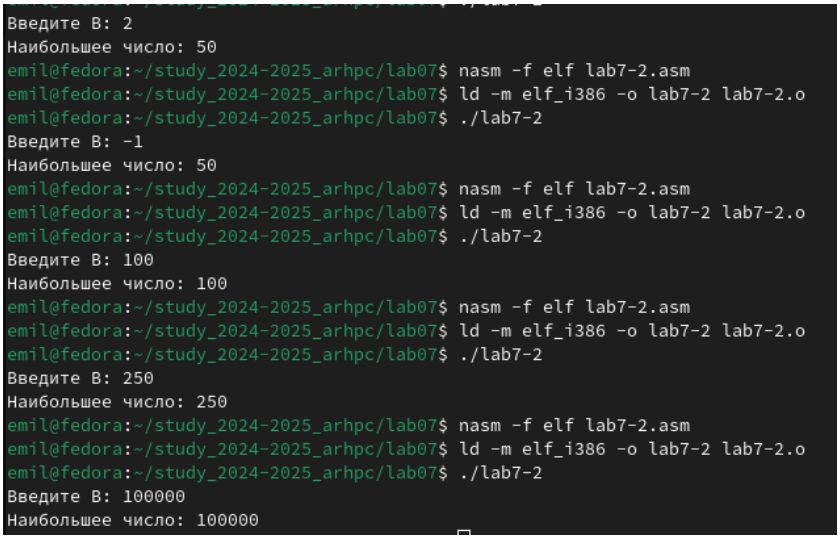


Рис. 4.7: Запуск исполняемого файла

4.3 Изучение структуры файлы листинга

Создаю файл листинга для программы из файла

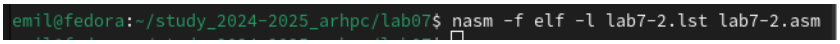


Рис. 4.8: Запуск исполняемого файла

Открываю его через mcedit

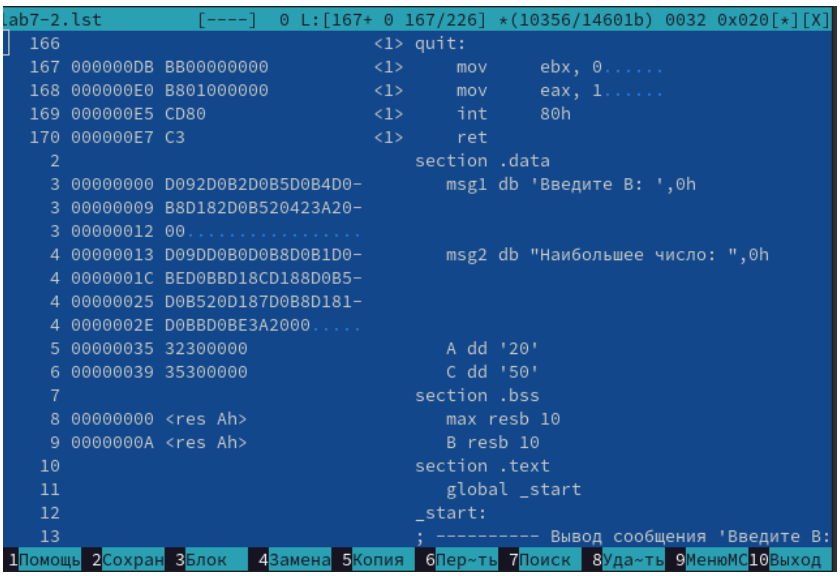


Рис. 4.9: Открытие файла в mcedit

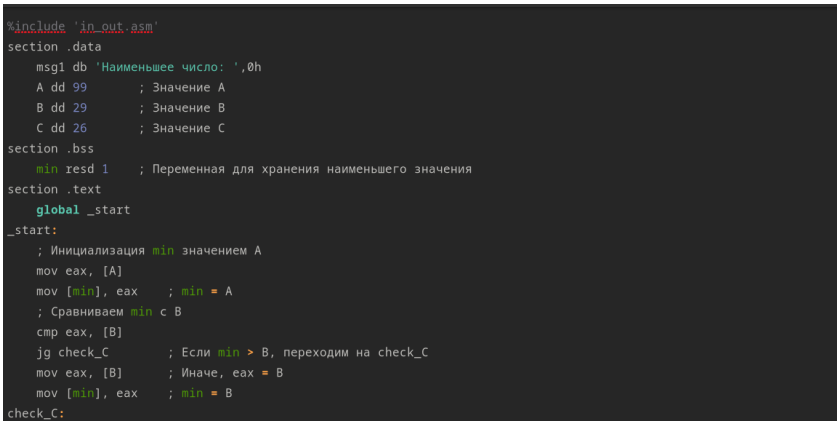
При компиляции и сборке программы на ассемблере создаются следующие файлы: Объектный файл (.o): Это промежуточный файл, содержащий машинный код, но ещё не готовый для выполнения. Исполняемый файл: После связывания объектных файлов с библиотеками (например, с помощью ld), создается исполняемый файл, который можно запустить. Файл листинга (.lst): Это текстовый файл, который включает исходный код программы вместе с адресами и скомпилированным машинным кодом. В этом файле обычно содержатся комментарии и информация о процессе компиляции.

В файл листинга могут быть добавлены следующие элементы: Исходный код: Полный исходный код программы, как он написан в ассемблере. Адреса: Для каждой инструкции будут указаны адреса в памяти, по которым эти инструкции будут располагаться после компиляции. Машинный код: Бинарный код, соответствующий каждой инструкции, представленный в шестнадцатеричном формате. Комментарии: Комментарии из исходного кода, которые могут помочь понять логику программы. Информация о секциях: Данные о том, как разделены секции кода (.text, .data, .bss и т.д.) и их размеры. Ошибки и предупреждения: Если при компиляции были обнаружены ошибки или предупреждения, они также могут

быть записаны в файл листинга.

5 Выполнение заданий для самостоятельной работы

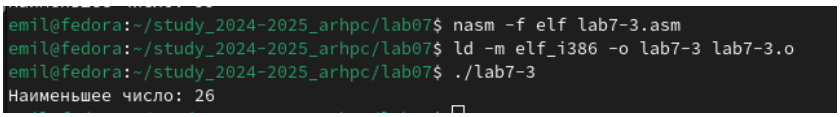
Создаю файл с названием lab7-3.asm, написал программу для нахождения наименьшего из 3 переменных,значения переменных беру исходя из своего варианта, полученного в ходе лабораторной работы номер 6,номер моего варианта 12.



```
%include 'in_out.asm'
section .data
    msg1 db 'Наименьшее число: ',0h
    A dd 99       ; Значение A
    B dd 29       ; Значение B
    C dd 26       ; Значение C
section .bss
    min resd 1    ; Переменная для хранения наименьшего значения
section .text
    global _start
_start:
    ; Инициализация min значением A
    mov eax, [A]
    mov [min], eax    ; min = A
    ; Сравниваем min с B
    cmp eax, [B]
    jg check_C        ; Если min > B, переходим на check_C
    mov eax, [B]      ; Иначе, eax = B
    mov [min], eax    ; min = B
check_C:
```

Рис. 5.1: Редактирование файла

Проверяю работу программы,программа работает верно.



```
emil@fedora:~/study_2024-2025_arhpc/lab07$ nasm -f elf lab7-3.asm
emil@fedora:~/study_2024-2025_arhpc/lab07$ ld -m elf_i386 -o lab7-3 lab7-3.o
emil@fedora:~/study_2024-2025_arhpc/lab07$ ./lab7-3
Наименьшее число: 26
```

Рис. 5.2: Запуск исполняемого файла

Создаю файл с названием lab7-4.asm, написал программу для вычисления

f(x),пишу программу для функции исходя из своего варианта, полученного в ходе лабораторной работы номер 6,номер моего варианта 12.

```
%include 'in_out.asm'

section .data
    prompt_x db 'Введите x: ', 0
    prompt_a db 'Введите a: ', 0
    result_msg db 'Результат f(x) = ', 0

section .bss
    x resb 10
    a resb 10
    result resb 10

section .text
    global _start

_start:
    ; ---------- Вывод сообщения 'Введите x: '
    mov eax, prompt_x
    call sprint
```

Рис. 5.3: Редактирование файла

```
emil@fedora:~/study_2024-2025_arhpc/lab07$ touch lab7-4.asm
emil@fedora:~/study_2024-2025_arhpc/lab07$ nasm -f elf lab7-4.asm
emil@fedora:~/study_2024-2025_arhpc/lab07$ ld -m elf_i386 -o lab7-4 lab7-4.o
emil@fedora:~/study_2024-2025_arhpc/lab07$ ./lab7-4
Введите x: 3
Введите a: 7
Результат f(x) = 21
emil@fedora:~/study_2024-2025_arhpc/lab07$ nasm -f elf lab7-4.asm
emil@fedora:~/study_2024-2025_arhpc/lab07$ ld -m elf_i386 -o lab7-4 lab7-4.o
emil@fedora:~/study_2024-2025_arhpc/lab07$ ./lab7-4
Введите x: 6
Введите a: 4
Результат f(x) = 1
```

Рис. 5.4: Запуск исполняемого файла

Программа работает верно,это я выяснил подставив значения.

6 Выводы

В результате выполнения данной лабораторной работы, я изучил команды условного и безусловного переходов, приобрел навыки написания программ с использованием переходов и познакомился с назначением и структурой файла листинга