|  |  |
| --- | --- |
| Gerb-BMSTU_01 | ***«*Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ \_\_\_\_\_\_\_ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ\_\_\_\_\_\_\_\_

КАФЕДРА \_\_\_\_\_\_\_\_\_КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**ОТЧЕТ**

**по лабораторной работе № 3**

**Дисциплина:** Машинно-зависимые языки и основы компиляции

**Название лабораторной работы:** Программирование ветвлений и итерационных циклов

Студент гр. ИУ6 - 42Б **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Г. Д. Нефедов**

(Группа) (Подпись, дата) (И.О. Фамилия)

Преподаватель  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(Подпись, дата) (И.О. Фамилия)

Москва, 2024

**Цель работы:** изучение средств и приемов программирования ветвлений и итерационных циклов на языке ассемблера.

Задание:

Вычислить целочисленное выражение (смотреть ниже рисунок 1):

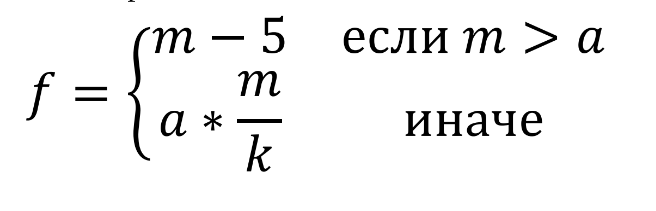
 Рисунок 1 — Задание

Схема алгоритма представлена ниже на рисунке 2.

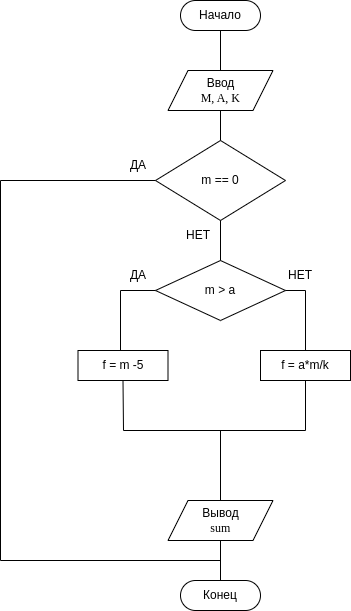


Рисунок 2 — Схема алгоритма

Код программы

%include "lib.asm"

section .data

ib db "Enter M, A, K: ", 0

ia db "Your result: ", 0

section .bss

outbuf resb 10

lenout equ $-outbuf

M resw 1

A resw 1

K resw 1

inbuf resb 10

lenIn equ $-inbuf

msgB resb 15

lenMsgB equ $-msgB

msgA resb 13

lenMsgA equ $-msgA

section .text

global \_start

\_start:

jmp InputM

InputM:

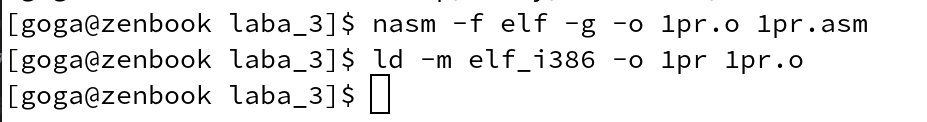
;; вывовдим сообщение ib

mov eax, 4

mov ebx, 1

mov ecx, ib

Выполним трансляцию и компоновку программы, как показано ниже на рисунке 3.

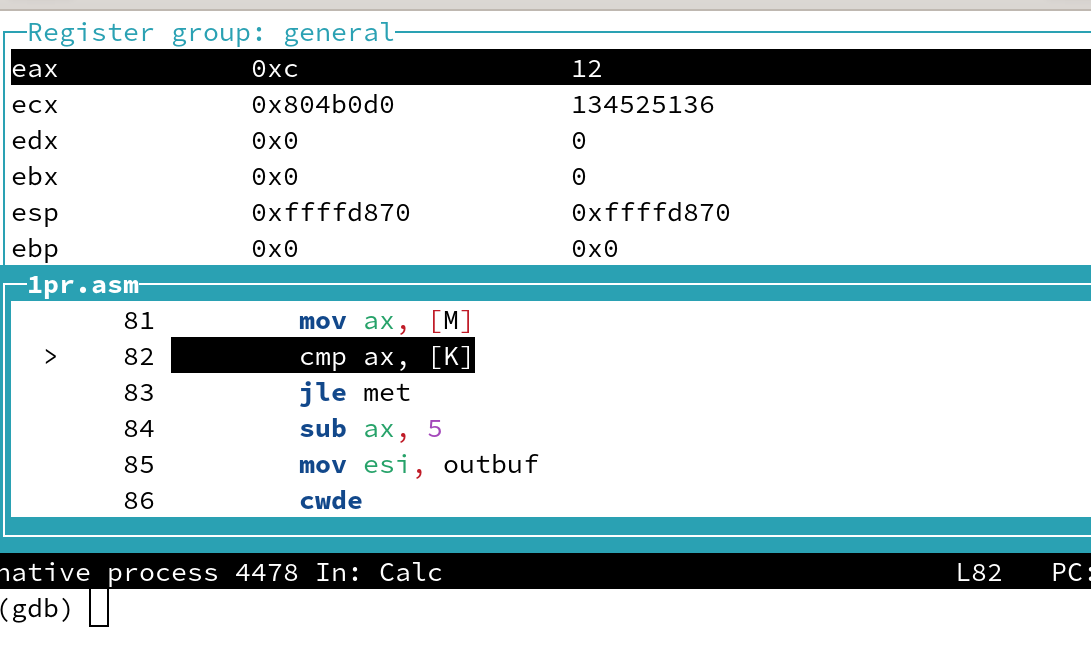
Рисунок 3 — Трансляция и компановка программы

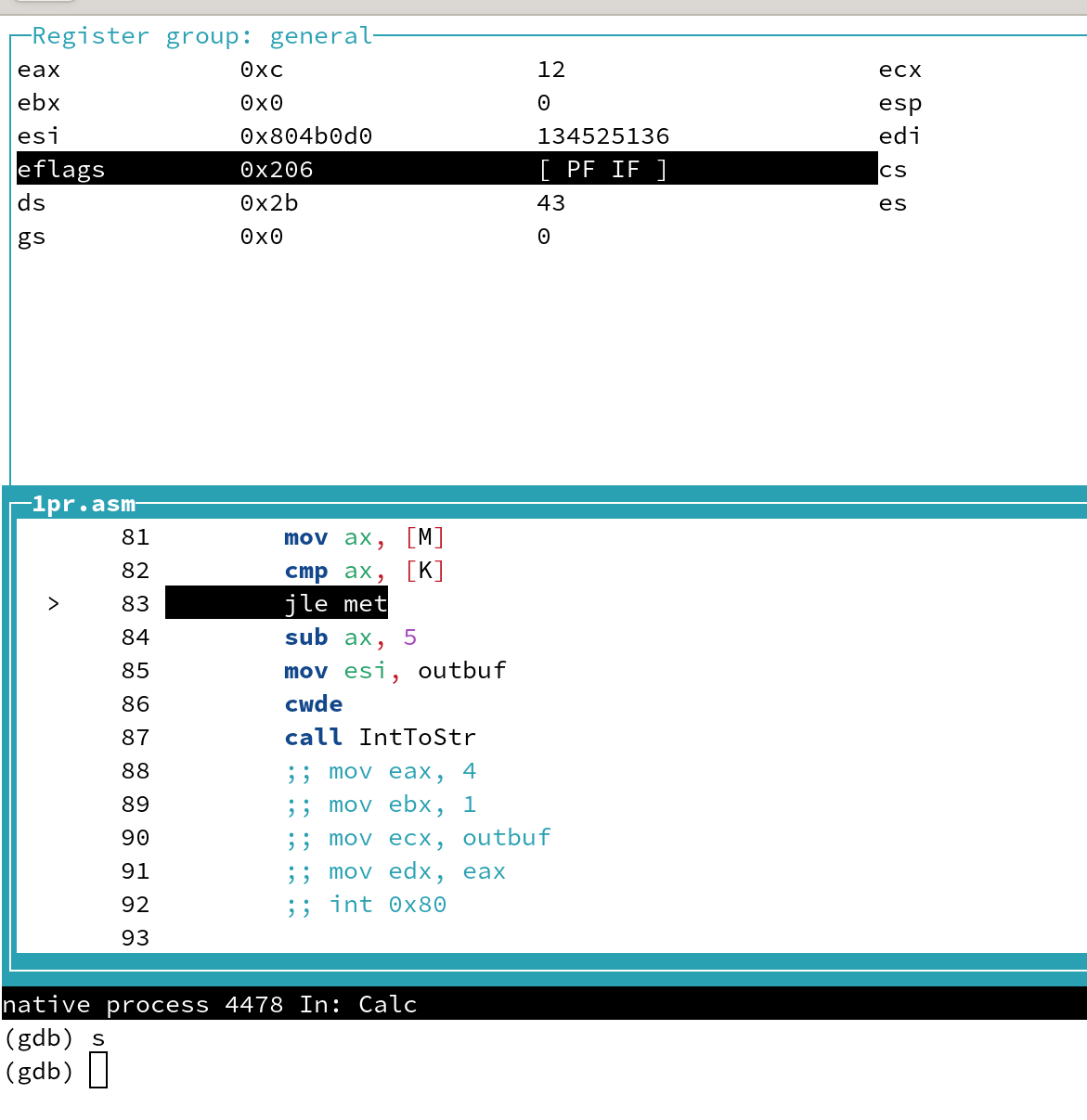
|  |
| --- |
| %include "lib.asm"  section .data  ib db "Enter M, A, K: ", 0  ia db "Your result: ", 0  ir db "Error",0  section .bss  outbuf resb 10  lenout equ $-outbuf  M resw 1  A resw 1  K resw 1  inbuf resb 10  lenIn equ $-inbuf  msgB resb 15  lenMsgB equ $-msgB  msgR resb 15  lenMsgR equ $-msgR  msgA resb 13  lenMsgA equ $-msgA  section .text  global \_start  \_start:  jmp InputM  InputM:  ;; вывовдим сообщение ib  mov eax, 4  mov ebx, 1  mov ecx, ib  mov edx, lenMsgB  int 0x80  ;; считываем введеное число  mov eax, 3  mov ebx, 0  mov ecx, inbuf  mov edx, 10  int 0x80  ;; передаем параметры в функцию  mov esi, inbuf  call StrToInt  cmp EBX, 0  mov [M], ax  jmp InputA  InputA:  ;; считываем введеное число  mov eax, 3  mov ebx, 0  mov ecx, inbuf  mov edx, 10  int 0x80  ;;передаем параметры в функцию  mov esi, inbuf  call StrToInt  cmp EBX, 0  mov [A], ax  jmp InputK  InputK:  ;; считываем введеное число  mov eax, 3  mov ebx, 0  mov ecx, inbuf  mov edx, 10  int 0x80  ;;передаем параметры в функцию  mov esi, inbuf  call StrToInt  cmp EBX, 0  cmp ax, 0  jz error  mov [K], ax  jmp Calc  Calc:  mov ax, [M]  cmp ax, [A]  jle met  sub ax, 5  mov esi, outbuf  cwde  call IntToStr  int 0x80  jmp exit  met:  mov dx, [A]  imul ax, dx  mov cx, [K]  cwd  idiv cx  mov esi, outbuf  cwde  call IntToStr  jmp exit  error:  mov eax, 4  mov ebx, 1  mov ecx, ir  mov edx, lenMsgR  int 0x80  ;; выводим результат  mov eax, 1  xor ebx, ebx  int 0x80  exit:  mov eax, 4  mov ebx, 1  mov ecx, ia  mov edx, lenMsgA  int 0x80  ;; выводим результат  mov eax, 4  mov ebx, 1  mov ecx, outbuf  mov edx, eax  int 0x80  mov eax, 1  xor ebx, ebx  int 0x80 |

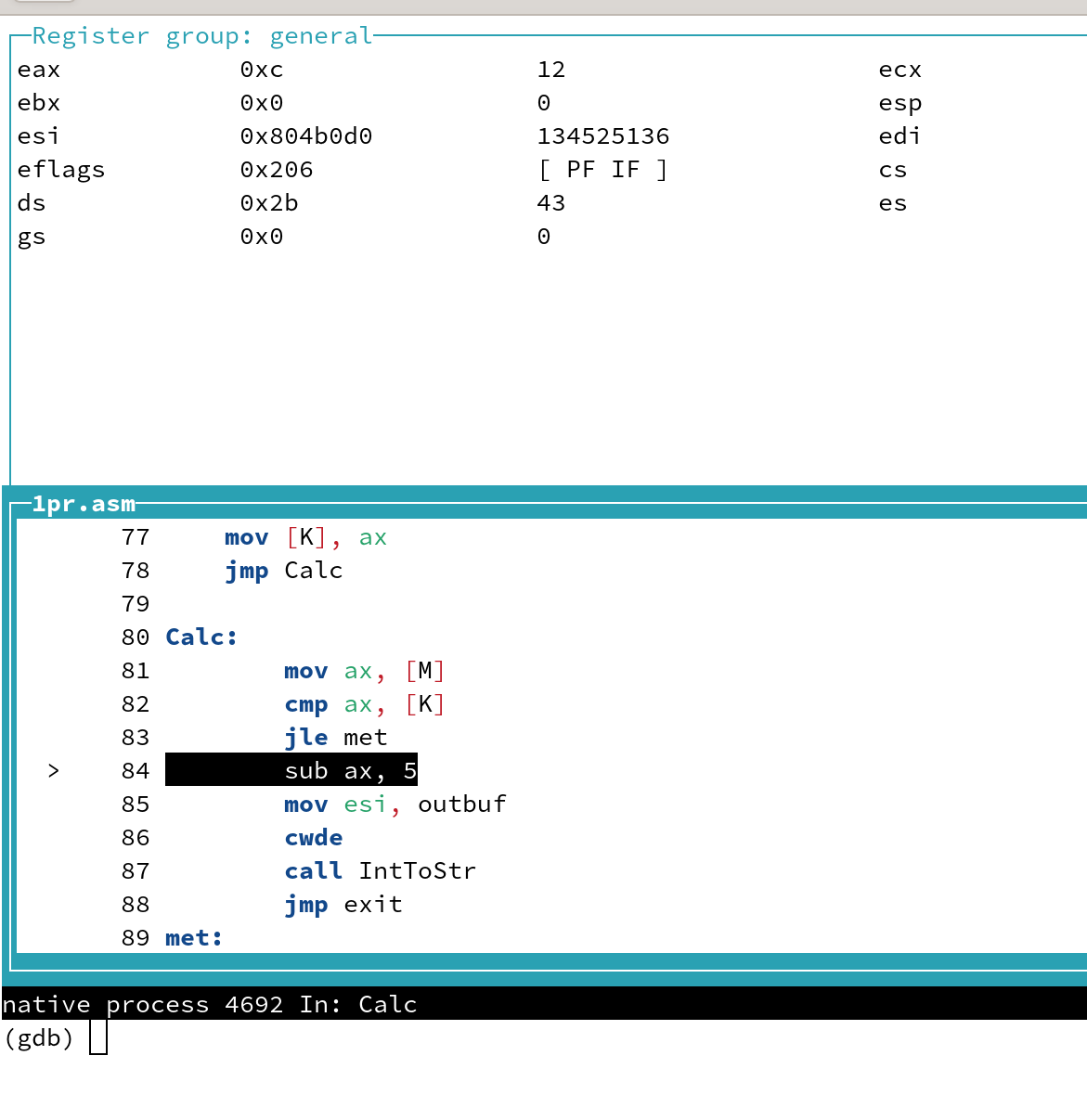
Отладка представлены на рисунках 4 -13:

Для отладки m > a возьмем числа : m = 12; a = 7; k = 9

Теперь отладим программу, как показано ниже на рисунках 4, 5, 6, 7

Рисунок 4 — Запись m в регистр ax

Рисунок 5 — Сравниваем ax, K

Рисунок 6 — m > a (не переходим на метку met)

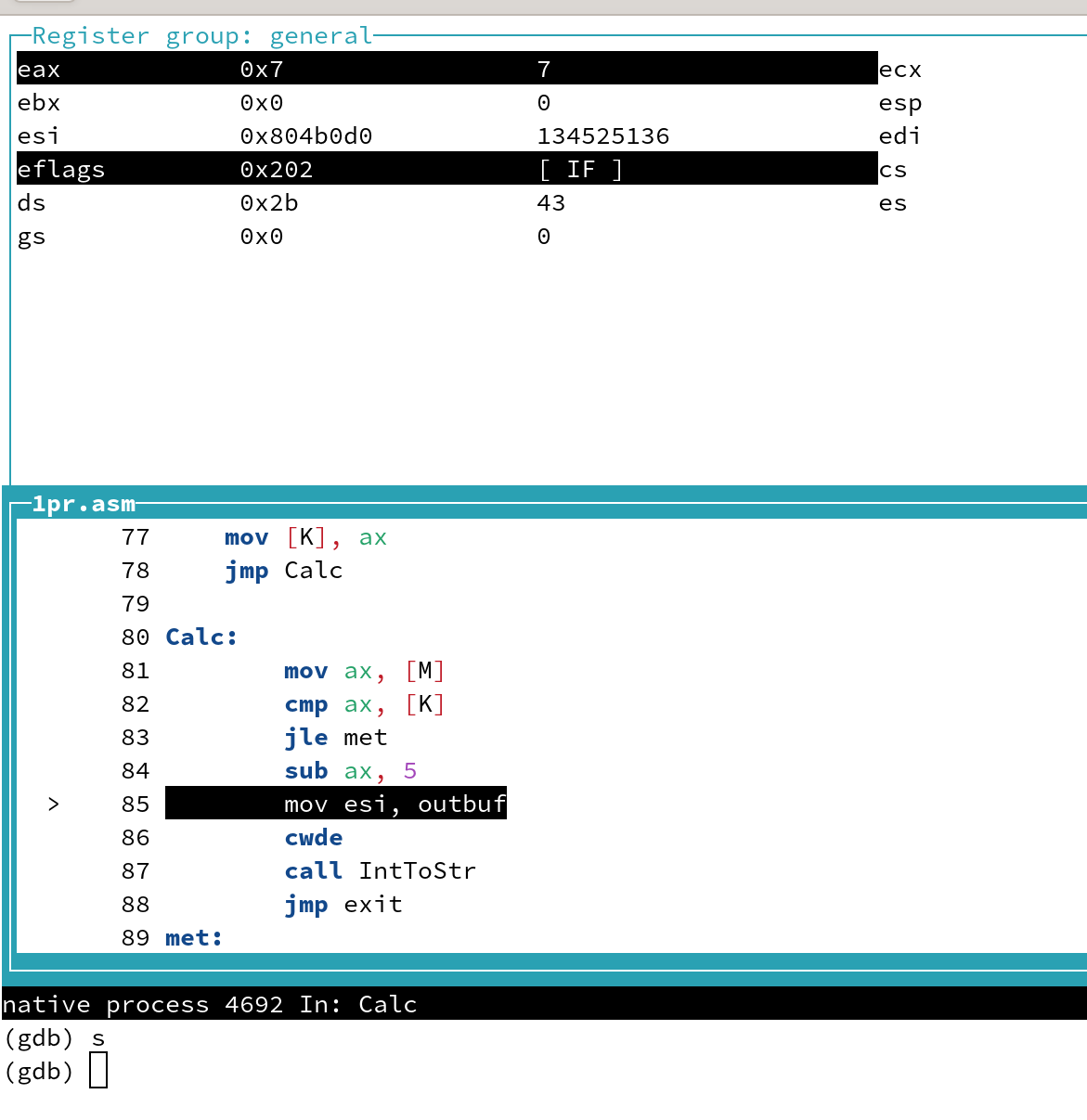


Рисунок 7 — Вычитаем из M пять

Для отладки m < a возьмем числа : m = 15; a = 20; k = 5

Теперь отладим программу, как показано ниже на рисунках

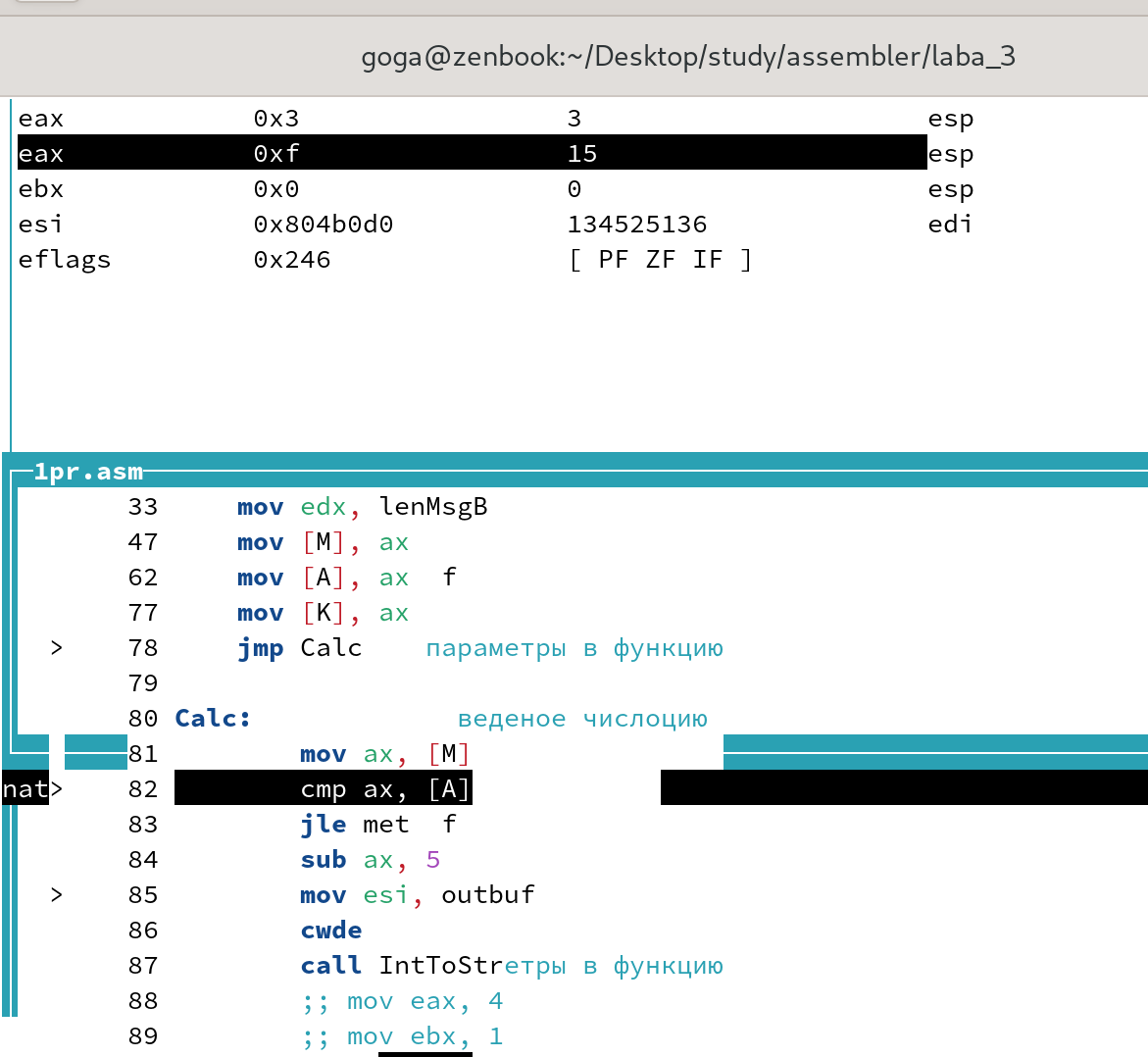


Рисунок 8 — Запись m в регистр ax

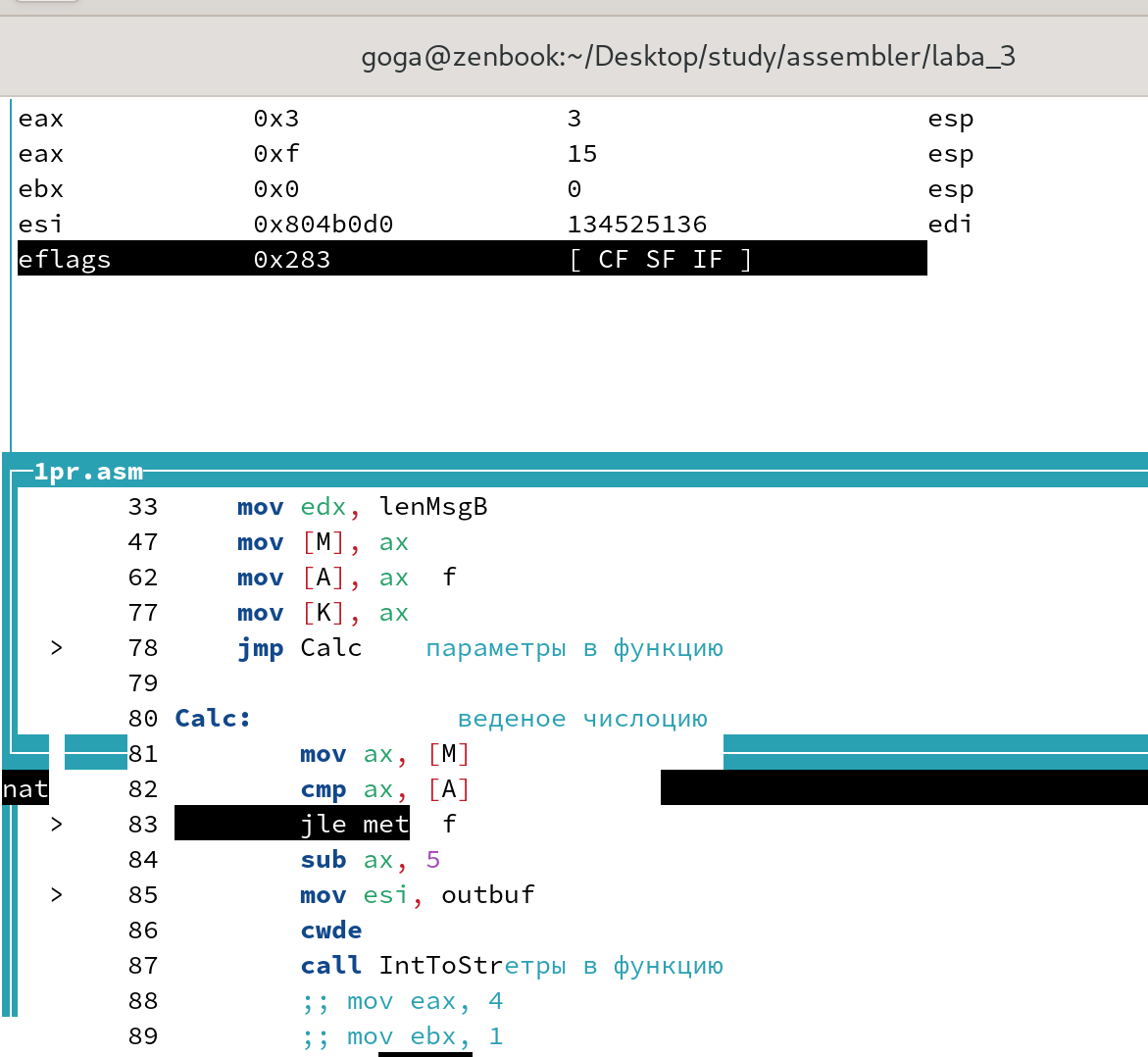
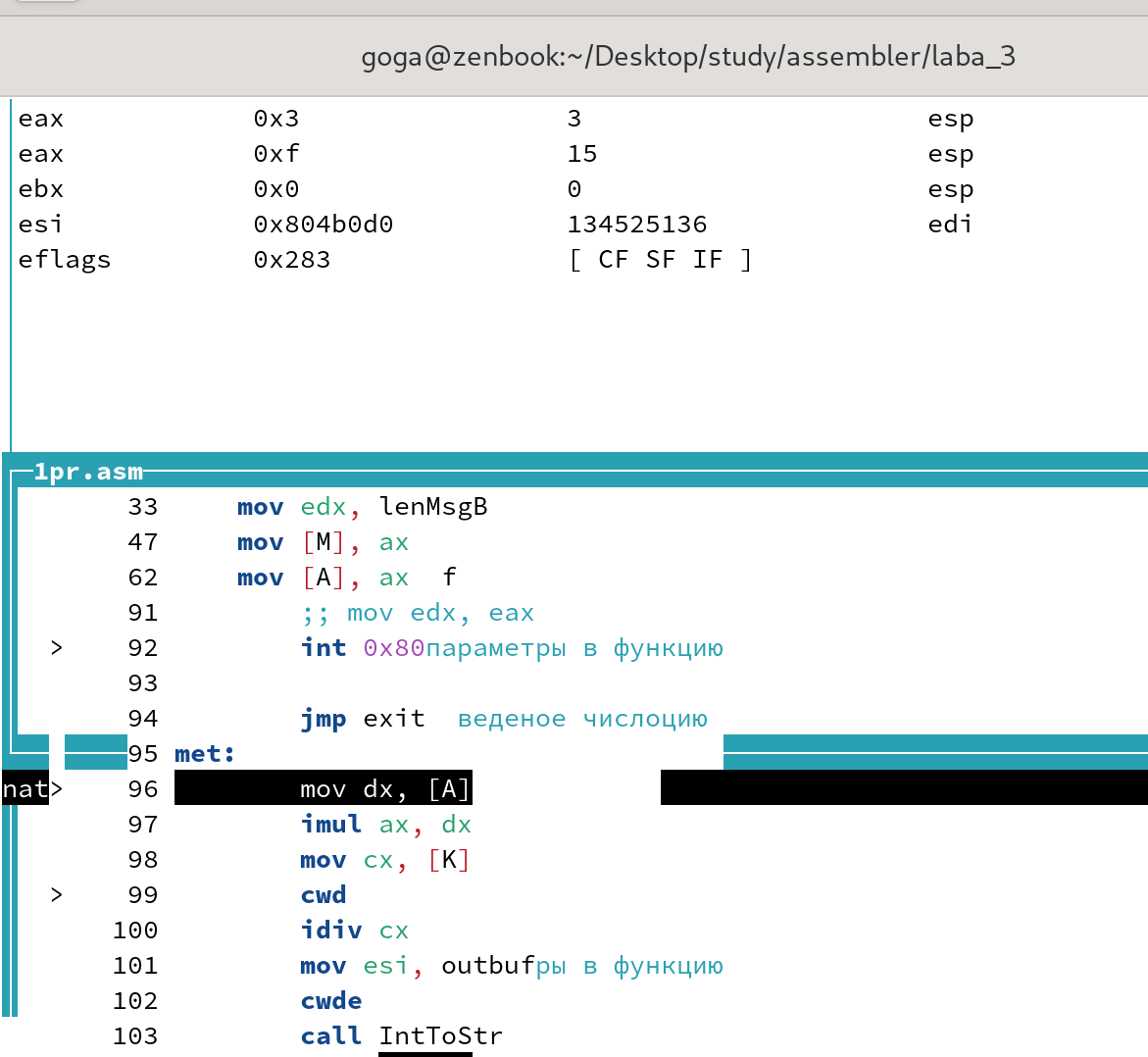
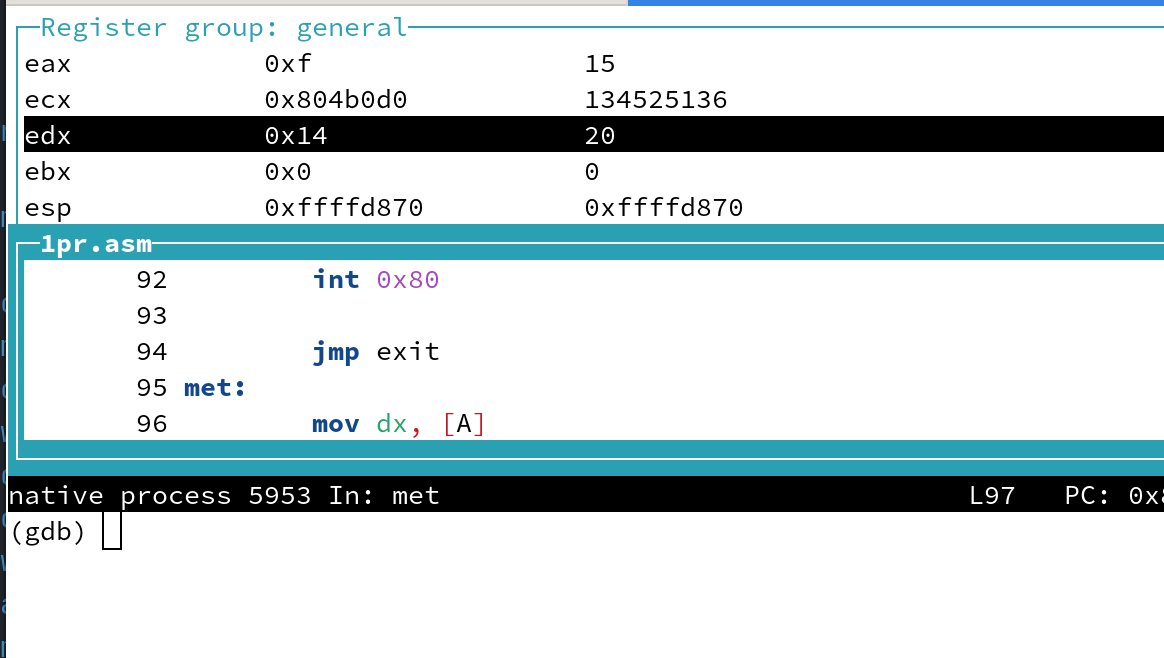
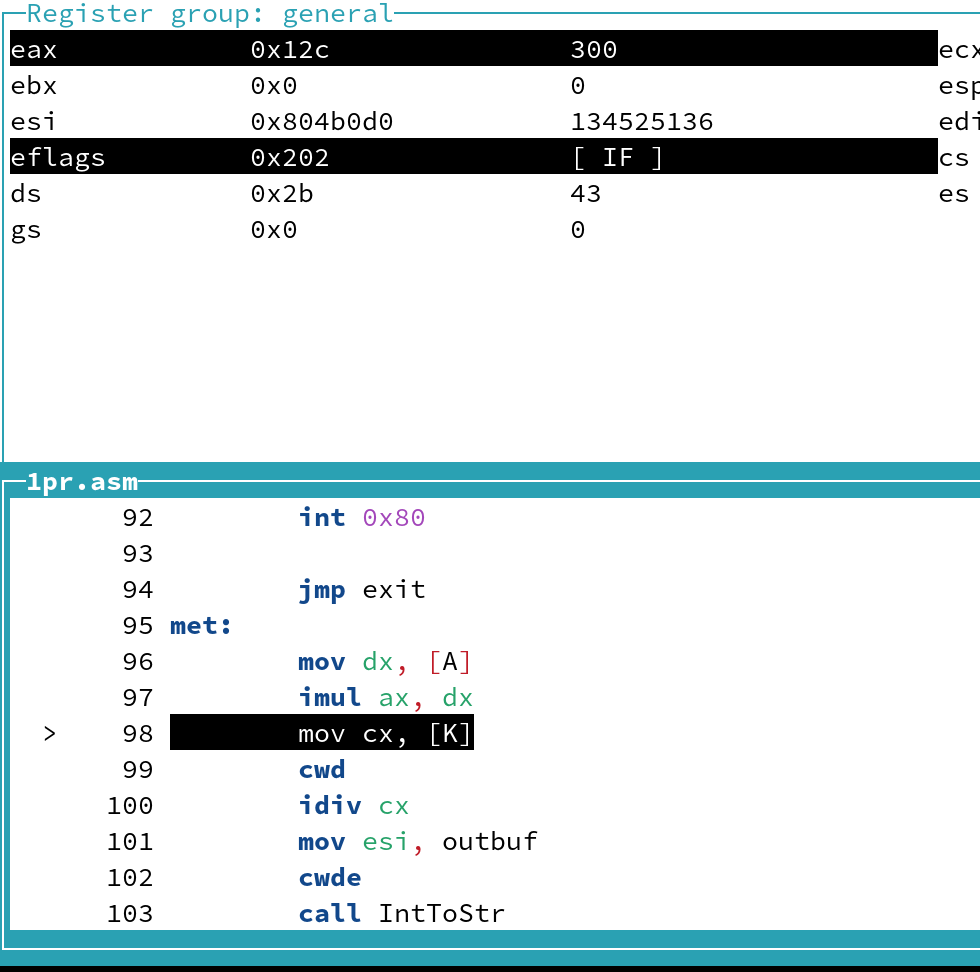
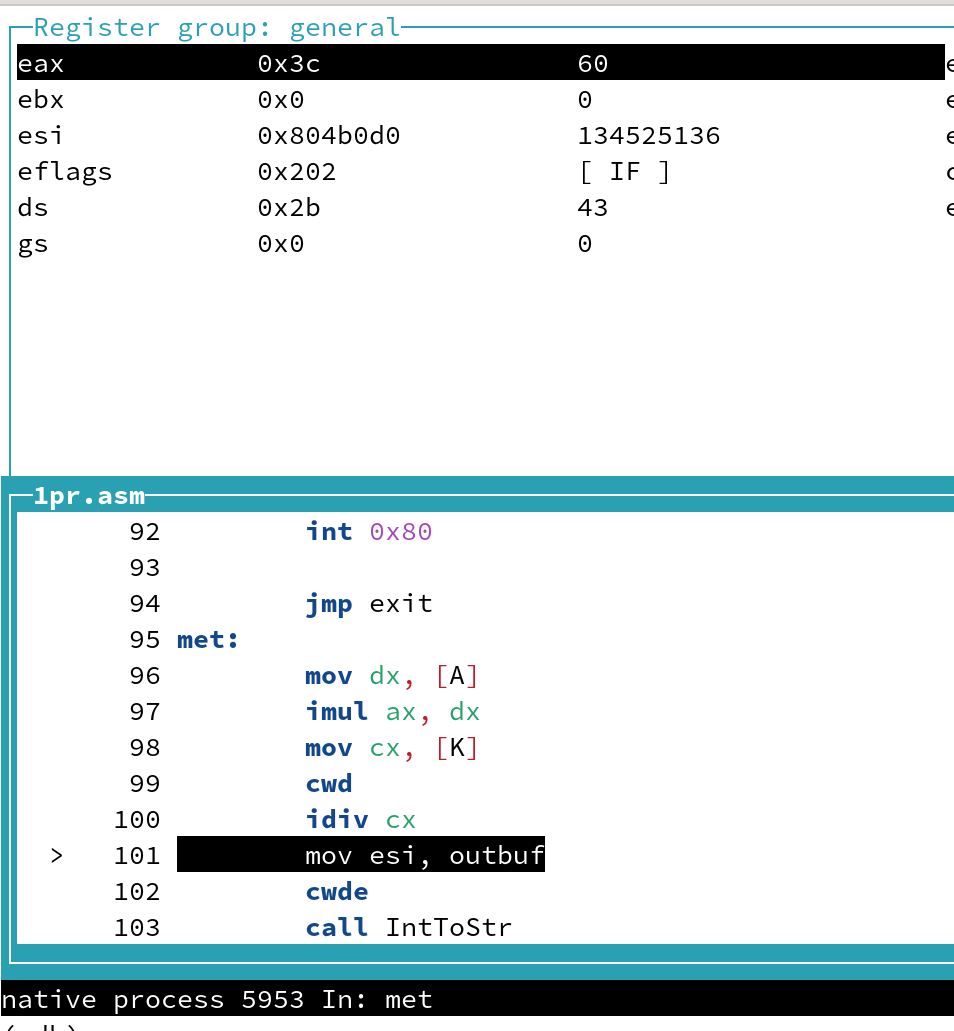


Рисунок 9 — Сравниваем ax, A

Рисунок 10 — Переходим на метку

Рисунок 11 — Записали в edx значение A

Рисунок 12 — Умножение A на M

Рисунок 13 — idiv cx

Тестирование

|  |  |  |
| --- | --- | --- |
| Исходные данные | Ожидаемый результат | Полученный результат |
| M = 25 ; A = 3 ;K = 7 | 20 | 20 |
| M = -10; A = -15 ; K = -3 | -15 | -15 |
| M = 225; A = 0 ; K = 22 | 220 | 220 |
| M = 20; A = 20 ; K = 5 | 80 | 80 |
| M = -5; A = 10 ; K = 5 | -10 | -10 |

Вывод: В процессе выполения лабораторной работы была разобрана программа создания условных операторов в ассемблере, а так же написана и протестирована программа с простейшим ветвлением.

**Контрольные вопросы**

**1.**Какие машинные команды используют при программировании ветвлений и циклов?

a. Условные переходы (Branching / Conditional Jumps):

1. JE (Jump if Equal) - Эта команда перехода выполняется, если два значения равны. Например, если результат сравнения двух чисел или значений в регистрах равен друг другу, то выполнение программы переходит к указанному адресу.
2. JNE (Jump if Not Equal) - Эта команда перехода выполняется, если два значения не равны. Если результат сравнения двух значений не равен друг другу, то управление программой переходит к заданному адресу.
3. JZ (Jump if Zero) - Эта команда перехода выполняется, если значение ноль. Обычно используется после выполнения арифметических или логических операций, чтобы проверить, равен ли результат нулю.
4. JNZ (Jump if Not Zero) - Эта команда перехода выполняется, если значение не равно нулю. Она используется для перехода в случае, если результат операции не равен нулю.
5. JG (Jump if Greater) - Эта команда перехода выполняется, если первое значение больше второго. Если первое значение больше второго, то управление программой переходит к указанному адресу.
6. JGE (Jump if Greater or Equal) - Эта команда перехода выполняется, если первое значение больше или равно второму. Если первое значение больше или равно второму, то управление программой переходит к указанному адресу.
7. JL (Jump if Less) - Эта команда перехода выполняется, если первое значение меньше второго. Если первое значение меньше второго, то управление программой переходит к указанному адресу.
8. JLE (Jump if Less or Equal) - Эта команда перехода выполняется, если первое значение меньше или равно второму. Если первое значение меньше или равно второму, то управление программой переходит к указанному адресу.

b. Безусловные переходы (Unconditional Jumps):

1. JMP (Jump) - Это безусловный переход, который просто переводит выполнение программы к указанному адресу, независимо от каких-либо условий.

c. Циклы (Loops):

1. LOOP - Эта команда используется для реализации цикла. Она уменьшает счетчик цикла (обычно регистр CX), и если он не равен нулю, программа выполняет переход к началу цикла для повторного выполнения кода внутри него.

2. Выделите в своей программе фрагмент, реализующий ветвление. Каково назначение каждой машинной команды фрагмента?

Calc: mov ax, [M] ; Загрузка значения M в регистр

AX cmp ax, [A] ; Сравнение значения M с значением A

jle met ; Если M меньше или равно A, выполнить переход к метке met

sub ax, 5 ; Вычитание 5 из значения M, если оно больше A

mov esi, outbuf ; Помещение указателя на буфер вывода в регистр ESI

cwde ; Расширение знака для регистра AX до EAX

call IntToStr ; Вызов функции для преобразования целого числа в строку

int 0x80 ; Прерывание для вывода строки

jmp exit ; Безусловный переход к метке exit met

mov dx, [A] ; Загрузка значения A в регистр DX

imul ax, dx ; Умножение значения M на значение A

mov cx, [K] ; Загрузка значения K в регистр CX cwd ; Расширение знака для регистра AX до DX:AX

idiv cx ; Деление DX:AX на значение K

mov esi, outbuf ; Помещение указателя на буфер вывода в регистр ESI

cwde ; Расширение знака для регистра AX до EAX

call IntToStr ; Вызов функции для преобразования целого числа в строку

jmp exit ; Безусловный переход к метке exit

**3. Чем вызвана необходимость использования команд безусловной передачи управления?**

Необходимость использования команд безусловной передачи управления возникает из-за необходимости изменения потока выполнения программы в зависимости от различных условий или требований.

**Завершение программы**: После завершения выполнения определенной части программы может потребоваться переход к завершающей части программы или к команде выхода из программы, которая завершает ее выполнение.

**Обработка ошибок**: Если возникает ошибка или неожиданное условие, которое требует немедленного завершения работы текущей части программы или выполнения определенных действий для обработки ошибки, может потребоваться использование команды безусловного перехода к соответствующему блоку кода

**Реализация циклов и ветвлений**: Для реализации циклов и ветвлений в программе могут использоваться команды безусловного перехода для определения места начала или завершения цикла, а также для перехода к различным ветвям исполнения в зависимости от условий.