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 (ИУ6)**

О т ч е т

по лабораторной работе № 3

Название: Программирование целочисленных вычислений

Дисциплина: Машинно-зависимые языки и основы компиляции

Студент гр. ИУ6-41Б 01.03.2023 Т. Е. Старжевский
(Подпись, дата) (И.О. Фамилия)

Преподаватель 01.03.2023 С. С. Данилюк
(Подпись, дата) (И.О. Фамилия)

Москва, 2023

Введение

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

Задачи работы:

1) Разработать схему алгоритма решения задачи.

2) Написать программу на языке ассемблера, которая вычисляет заданное выражение.

3) Протестировать.

Ход работы

Задание 1

Выражение, заданное 20-ому варианту показано на рисунке 1:

$$f = \begin{cases} \dfrac{3 * a * x}{b - 5} & \text{если } j < 5 \\ -12 & \text{иначе} \end{cases}$$

Рисунок 1 – заданное выражение

Схема алгоритма показана на рисунке 2:

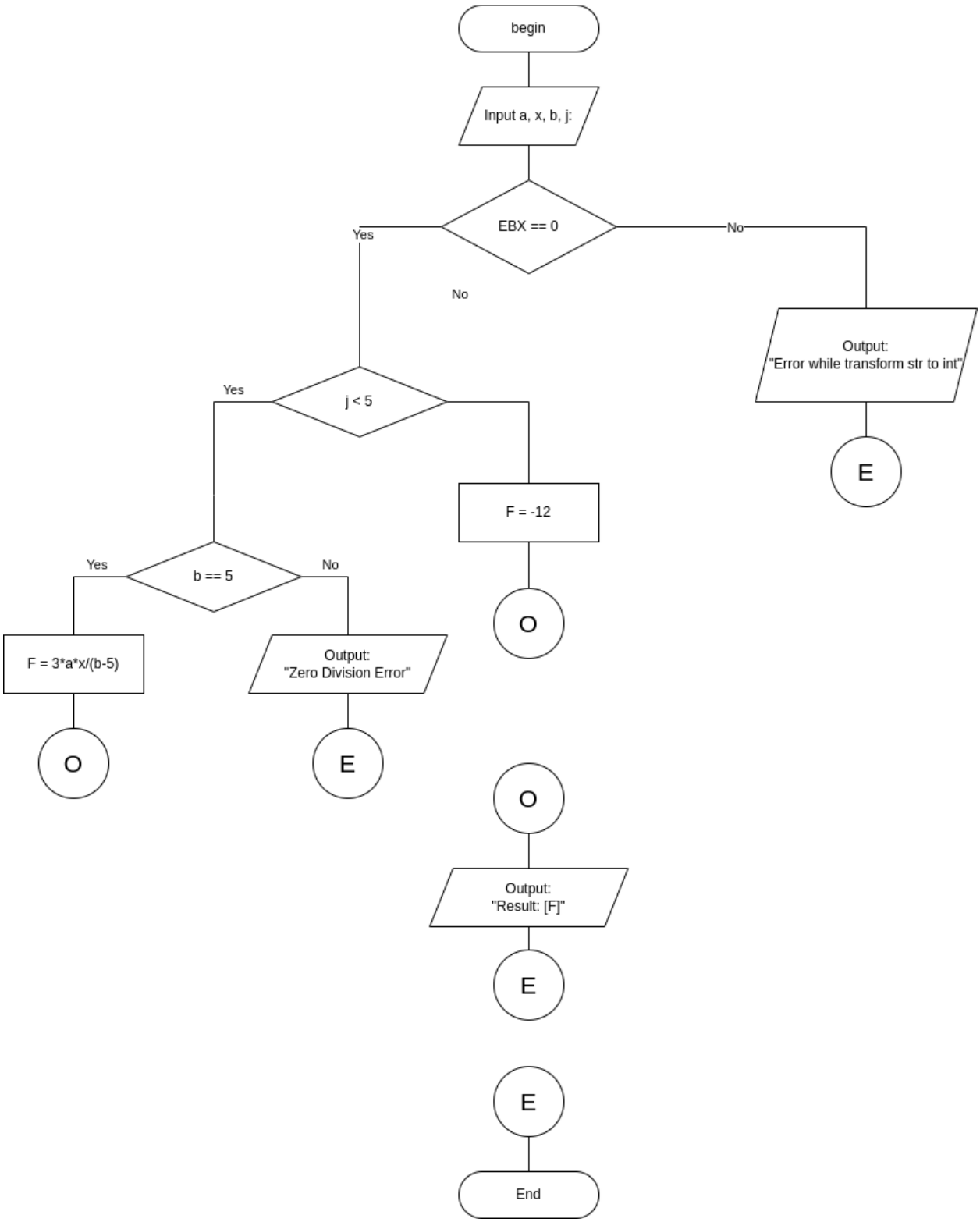

Рисунок 2 – схема алгоритма

Задание 2

Код программы, вычисляющий данное выражение:

```
%include "../lib64.asm"
section .data; сегмент инициализированных переменных
Hello1Msg dq "Input a: "
```

```
lenHello1 equ $-Hello1Msg
Hello2Msg dq "Input x: "
lenHello2 equ $-Hello2Msg
Hello3Msg dq "Input b: "
lenHello3 equ $-Hello3Msg
Hello4Msg dq "Input j: "
lenHello4 equ $-Hello4Msg
ErrorSTIMsg dq "Error while transform str to int", 10
lenErrorSTI equ $-ErrorSTIMsg
ZeroDivMsg dq "Zero Division Error", 10
lenZeroDiv equ $-ZeroDivMsg

ResMsg dq "Result: "
lenRes equ $-ResMsg

section .bss
InBuf resq 10
lenIn equ $-InBuf
OutBuf resq 10
lenOut equ $-OutBuf
a resq 1
x resq 1
b resq 1
j resq 1
F resq 1

section .text ; сегмент кода
global _start


_start:
; input
; Input a
mov rax, 1; системная функция 1 (write)
mov rdi, 1; дескриптор файла stdout=1
mov rsi, Hello1Msg ; адрес выводимой строки
mov rdx, lenHello1 ; длина строки
syscall; вызов системной функции
;end

; read data to InBuf
mov rax, 0; системная функция 0 (read)
mov rdi, 0 ; дескриптор файла stdout=0
lea rsi, InBuf ; передаем указатель на буфер
mov rdx, lenIn ; длина строки
syscall
; end

; InBuf To string
mov RSI, InBuf
```

```
call StrToInt64; Вход: ESI Выход: EAX, EBX содержит 0 if errors = 0
cmp EBX, 0
jne .STIError
mov [a], RAX
; end

; Input x
mov rax, 1; системная функция 1 (write)
mov rdi, 1; дескриптор файла stdout=1
mov rsi, Hello2Msg ; адрес выводимой строки
mov rdx, lenHello2 ; длина строки
syscall; вызов системной функции
;end

; read data to InBuf
mov rax, 0; системная функция 0 (read)
mov rdi, 0 ; дескриптор файла stdout=0
lea rsi, InBuf ; передаем указатель на буфер
mov rdx, lenIn ; длина строки
syscall
; end

; InBuf To string
mov RSI, InBuf
call StrToInt64; Вход: ESI Выход: EAX, EBX содержит 0 if errors = 0
cmp EBX, 0
jne .STIError
mov [x], RAX
; end

; Input b
mov rax, 1; системная функция 1 (write)
mov rdi, 1; дескриптор файла stdout=1
mov rsi, Hello3Msg ; адрес выводимой строки
mov rdx, lenHello3 ; длина строки
syscall; вызов системной функции
;end

; read data to InBuf
mov rax, 0; системная функция 0 (read)
mov rdi, 0 ; дескриптор файла stdout=0
lea rsi, InBuf ; передаем указатель на буфер
mov rdx, lenIn ; длина строки
syscall
; end

; InBuf To string
mov RSI, InBuf
call StrToInt64; Вход: ESI Выход: EAX, EBX содержит 0 if errors = 0
cmp EBX, 0
```

```
jne .STIError
mov [b], RAX
; end

; Input j
mov rax, 1; системная функция 1 (write)
mov rdi, 1; дескриптор файла stdout=1
mov rsi, Hello4Msg ; адрес выводимой строки
mov rdx, lenHello4 ; длина строки
syscall; вызов системной функции
;end

; read data to InBuf
mov rax, 0; системная функция 0 (read)
mov rdi, 0 ; дескриптор файла stdout=0
lea rsi, InBuf ; передаем указатель на буфер
mov rdx, lenIn ; длина строки
syscall
; end

; InBuf To string
mov RSI, InBuf
call StrToInt64; Вход: ESI Выход: EAX, EBX содержит 0 if errors = 0
cmp EBX, 0
jne .STIError
mov [j], RAX
; end
jmp .countResult
; end

.countResult:
mov rax, [j]
cmp rax, 5
jg .jtrue
mov rax, [b]
cmp rax, 5
je .ZeroDivError
mov rax, [x]
mov rbx, [a]
imul rbx
mov rbx, 3
imul rbx
mov rbx, [b]
sub rbx, 5
idiv rbx
mov [F], rax
jmp .output

.jtrue:
mov rax, -12
```

```
mov [F], rax
jmp .output

; end

.STIError:
mov rax, 1; системная функция 1 (write)
mov rdi, 1; дескриптор файла stdout=1
mov rsi, ErrorSTIMsg ; адрес выводимой строки
mov rdx, lenErrorSTI ; длина строки
syscall; вызов системной функции
jmp .end
;end

.ZeroDivError:
mov rax, 1; системная функция 1 (write)
mov rdi, 1; дескриптор файла stdout=1
mov rsi, ZeroDivMsg ; адрес выводимой строки
mov rdx, lenZeroDiv ; длина строки
syscall; вызов системной функции
jmp .end
;end

.output:
; Output
; Result to string
mov rsi, OutBuf
mov rax, [F]
cwde
call IntToStr64
; end
mov rax, 1; системная функция 1 (write)
mov rdi, 1; дескриптор файла stdout=1
mov rsi, ResMsg ; адрес выводимой строки
mov rdx, lenRes ; длина строки
syscall; вызов системной функции

mov rax, 1; системная функция 1 (write)
mov rdi, 1; дескриптор файла stdout=1
mov rsi, OutBuf ; адрес выводимой строки
mov rdx, lenOut ; длина строки
syscall; вызов системной функции
jmp .end
;end

.end:
; close program
mov rax, 60; системная функция 60 (exit)
xor rdi, rdi; return code 0
syscall; вызов системной функции
```

```
; end
```

Задание 3

Тестирование программы показаны на таблице 1:

Таблица 1 – Таблица тестирования

Исходные данные				Ожидаемый результат	Полученный результат
a	x	b	j	F	F
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	7	-12	-12
1	1	0	2	0	0
1	1	0	9	-12	-12
2	3	8	3	6	6
2	3	5	8	-12	-12
2	3	5	3	Error	Error

Вывод: написана программа на языке ассемблер, которая вычисляет одно из двух предложенных выражений в зависимости от введенных значений.

Контрольные вопросы:

1) Какие машинные команды используют при программировании ветвлений и циклов?

– Cmp – для сравнения переменных, значений и регистров;

– J<условие> “название1” – if в ассемблере, если условие выполняется, то продолжение работы программы после перехода к “название1”;

– Jmp “название2” – продолжение работы программы после перехода к “название2”;

– “название1/2”: – после чего программа продолжает работать.

Примеры условий в ассемблере:

– JZ – переход по "ноль";

– JE – переход по "равно";

– JNZ – переход по "не нуль";

– JNE – переход по "не равно";

– JL – переход по "меньше";

– JNG, JLE – переход по "меньше или равно ";

– JG – переход по "больше";

– JNL, JGE – переход по "больше или равно ";

– JA – переход по "выше" (беззнаковое “больше”);

– JNA, JBE – переход по "не выше"(беззнаковое “не больше”);

– JB – переход по "ниже" (беззнаковое “меньше”);

– JNB, JAE – переход по "не ниже" (беззнаковое “не меньше”).

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

```
.countResult:
    mov rax, [j]; Поместили j в регистр rax
    cmp rax, 5; Сравнили с 5
    jg .jtrue; Если больше, то продолжаем работу программы с .jgtrue
    mov rax, [b]; Поместили b в регистр rax
    cmp rax, 5; Сравнили с 5
    je .ZeroDivError; Если равно, то переходим к ошибки деления на ноль
    mov rax, [x]; Поместили x
    mov rbx, [a]; Поместили a
    imul rbx; Умножили x * a
    mov rbx, 3; Поместили 3
    imul rbx; Умножили 3 * x * a
    mov rbx, [b]; Поместили b в rbx
    sub rbx, 5; rbx = b - 5
    idiv rbx; rax = 3 * x * a / b - 5
    mov [F], rax; Инициализировали результат
    jmp .output; Выводим результат
```

3) Чем вызвана необходимость использования команд безусловной передачи управления?

Когда проходит “if” и условие не выполняются, то программа продолжает работать дальше. И чтобы условие else не выполнилось, нужно этот фрагмент кода “перепрыгнуть” и тогда используются безусловные переходы.

4) Поясните последовательность команд, выполняющих операции ввода-вывода в вашей программе. Чем вызвана сложность преобразований данных при выполнении операций ввода-вывода?

Ввод переменных a, x, b, j. Вывод сообщений об ошибке или вывод результата.

При вводе каждая переменная преобразуется из строки в число и записывается в нужную переменную. Это делается в 15 строк кода для каждой: Вывод приветственного сообщения, считывание переменной, преобразование из строки в число.

Если ошибок не произошло, то нужно вывести результат. Он выводится в 10 строк: сначала преобразуем из строки его в число, потом выводим системным вызовом.