Отчёт по лабораторной работе №6

Петлин Артём Дмитриевич

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

Освоение арифметических инструкций языка ассемблера NASM.

2 Задание для самостоятельной работы

1. Написать программу вычисления выражения y = f(x). Программа должна выводить выражение для вычисления, выводить запрос на ввод значения x, вычислять заданное выражение в зависимости от введенного x, выводить результат вычислений. Вид функции f(x) выбрать из таблицы 6.3 вариантов заданий в соответствии с номером полученным при выполнении лабораторной работы. Создайте исполняемый файл и проверьте его работу для значений x1 и x2 из 6.3.

Таблица 6.3. Выражения для $f(x)$ для задания №1

Номер варианта	Выражение для $f(x)$	x_1	x_2
1	$(10 + 2x)/3$	1	10
2	$(12x + 3)5$	1	6
3	$(2 + x)^2$	2	8
4	$\frac{4}{3}(x - 1) + 5$	4	10
5	$(9x - 8)/8$	8	64
6	$x^3/2 + 1$	2	5
7	$5(x - 1)^2$	3	5
8	$(11 + x) \cdot 2 - 6$	1	9
9	$10 + (31x - 5)$	3	1
10	$5(x + 18) - 28$	2	3
11	$10(x + 1) - 10$	1	7
12	$(8x - 6)/2$	1	5
13	$(8x + 6) \cdot 10$	1	4
14	$(\frac{x}{2} + 8) \cdot 3$	1	4
15	$(5 + x)^2 - 3$	5	1
16	$(10x - 5)^2$	3	1
17	$18(x + 1)/6$	3	1
18	$3(x + 10) - 20$	1	5
19	$(\frac{1}{3}x + 5) \cdot 7$	3	9
20	$x^3 \cdot \frac{1}{3} + 21$	1	3

При выполнении задания преобразовывать (упрощать) выражения для f(x) нельзя. При выполнении деления в качестве результата можно использовать только целую часть от деления и не учитывать остаток (т.е. 5 : 2 = 2).

3 Теоретическое введение

3.1 Адресация в NASM

Большинство инструкций на языке ассемблера требуют обработки операндов. Адрес операнда предоставляет место, где хранятся данные, подлежащие обработке. Это могут быть данные хранящиеся в регистре или в ячейке памяти. Далее рассмотрены все существующие способы задания адреса хранения операндов – способы адресации.
Существует три основных способа адресации:

- Регистровая адресация – операнды хранятся в регистрах и в команде используются имена этих регистров, например: mov ax,bx.

- Непосредственная адресация – значение операнда задается непосредственно в команде, Например: mov ax,2.

- Адресация памяти – операнд задает адрес в памяти. В команде указывается символическое обозначение ячейки памяти, над содержимым которой требуется выполнить операцию.

Например, определим переменную intg DD 3 – это означает, что задается область памяти размером 4 байта, адрес которой обозначен меткой intg. В таком случае, команда

```
mov eax,[intg]
```

копирует из памяти по адресу intg данные в регистр eax. В свою очередь команда

```
mov [intg],eax
```

запишет в память по адресу intg данные из регистра eax.
Также рассмотрим команду

```
mov eax,intg
```

В этом случае в регистр eax запишется адрес intg. Допустим, для intg выделена память начиная с ячейки с адресом 0x600144, тогда команда mov eax,intg аналогична команде mov eax,0x600144 – т.е. эта команда запишет в регистр eax число 0x600144.

3.2 Арифметические операции в NASM

3.2.1 Целочисленное сложение add.

Схема команды целочисленного сложения add (от англ. addition - добавление) выполняет сложение двух операндов и записывает результат по адресу первого операнда. Команда add работает как с числами со знаком, так и без знака и выглядит следующим образом:

```
add <операнд_1>, <операнд_2>
```

Допустимые сочетания операндов для команды add аналогичны сочетаниям операндов для команды mov.
Так, например, команда add eax,ebx прибавит значение из регистра eax к значению из регистра ebx и запишет результат в регистр eax.
Примеры:

```
add ax,5 ; AX = AX + 5
add dx,cx ; DX = DX + CX
add dx,cl ; Ошибка: разный размер операндов.
```

3.2.2 Целочисленное вычитание sub.

Команда целочисленного вычитания sub (от англ. subtraction – вычитание) работает аналогично команде add и выглядит следующим образом:
sub ,
Так, например, команда sub ebx,5 уменьшает значение регистра ebx на 5 и записывает результат в регистр ebx.

3.2.3 Команды инкремента и декремента.

Довольно часто при написании программ встречается операция прибавления или вычитания единицы. Прибавление единицы называется инкрементом, а вычитание — декрементом. Для этих операций существуют специальные команды: inc (от англ. increment) и dec (от англ. decrement), которые увеличивают и уменьшают на 1 свой операнд.
Эти команды содержат один операнд и имеет следующий вид:

```
inc <операнд>

dec <операнд>
```

Операндом может быть регистр или ячейка памяти любого размера. Команды инкремента и декремента выгодны тем, что они занимают меньше места, чем соответствующие команды сложения и вычитания.
Так, например, команда inc ebx увеличивает значение регистра ebx на 1, а команда inc ax уменьшает значение регистра ax на 1.

3.2.4 Команда изменения знака операнда neg.

Еще одна команда, которую можно отнести к арифметическим командам это команда изменения знака neg:

```
neg <операнд>
```

Команда neg рассматривает свой операнд как число со знаком и меняет знак операнда на противоположный. Операндом может быть регистр или ячейка памяти любого размера.

```
mov ax,1 ; AX = 1
neg ax ; AX = -1
```

3.2.5 Команды умножения mul и imul.

Умножение и деление, в отличии от сложения и вычитания, для знаковых и беззнаковых чисел производиться по-разному, поэтому существуют различные команды. Для беззнакового умножения используется команда mul (от англ. multiply – умножение):

```
mul <операнд>
```

Для знакового умножения используется команда imul:

```
imul <операнд>
```

Для команд умножения один из сомножителей указывается в команде и должен находиться в регистре или в памяти, но не может быть непосредственным операндом. Второй сомножитель в команде явно не указывается и должен находиться в регистре EAX,AX или AL, а результат помещается в регистры EDX:EAX, DX:AX или AX, в зависимости от размера операнда 6.1.

3.2.6 Команды деления div и idiv.

Для деления, как и для умножения, существует 2 команды div (от англ. divide - деление) и idiv:

```
div <делитель> ; Беззнаковое деление
idiv <делитель> ; Знаковое деление
```

В командах указывается только один операнд – делитель, который может быть регистром или ячейкой памяти, но не может быть непосредственным операндом. Местоположение делимого и результата для команд деления зависит от размера делителя. Кроме того, так как в результате деления получается два числа – частное и остаток, то эти числа помещаются в определённые регистры 6.2.
Например, после выполнения инструкций

```
mov ax,31
mov dl,15
div dl
```

результат 2 (31/15) будет записан в регистр al, а остаток 1 (остаток от деления 31/15) — в регистр ah.
Если делитель — это слово (16-бит), то делимое должно записываться в регистрах dx:ax. Так в результате выполнения инструкций

```
mov ax,2 ; загрузить в регистровую
mov dx,1 ; пару `dx:ax` значение 10002h
mov bx,10h
div bx
```

в регистр ax запишется частное 1000h (результат деления 10002h на 10h), а в регистр dx — 2 (остаток от деления).

3.3 Перевод символа числа в десятичную символьную запись

Ввод информации с клавиатуры и вывод её на экран осуществляется в символьном виде. Кодирование этой информации производится согласно кодовой таблице символов ASCII. ASCII – сокращение от American Standard Code for Information Interchange (Американский стандартный код для обмена информацией). Согласно стандарту ASCII каждый символ кодируется одним байтом. Расширенная таблица ASCII состоит из двух частей. Первая (символы с кодами 0-127) является универсальной (см. Приложение.), а вторая (коды 128-255) предназначена для специальных символов и букв национальных алфавитов и на компьютерах разных типов может меняться. Среди инструкций NASM нет такой, которая выводит числа (не в символьном виде). Поэтому, например, чтобы вывести число, надо предварительно преобразовать его цифры в ASCII-коды этих цифр и выводить на экран эти коды, а не само число. Если же выводить число на экран непосредственно, то экран воспримет его не как число, а как последовательность ASCII-символов – каждый байт числа будет воспринят как один ASCII-символ – и выведет на экран эти символы. Аналогичная ситуация происходит и при вводе данных с клавиатуры. Введенные данные будут представлять собой символы, что сделает невозможным получение корректного результата при выполнении над ними арифметических операций. Для решения этой проблемы необходимо проводить преобразование ASCII символов в числа и обратно. Для выполнения лабораторных работ в файле in_out.asm реализованы подпрограммы для преобразования ASCII символов в числа и обратно. Это:

- iprint – вывод на экран чисел в формате ASCII, перед вызовом iprint в регистр eax необходимо записать выводимое число (mov eax,).
- iprintLF – работает аналогично iprint, но при выводе на экран после числа добавляет к символ перевода строки.
- atoi – функция преобразует ascii-код символа в целое число и записает ре-

зультат в регистр eax, перед вызовом atoi в регистр eax необходимо записать число (mov eax,).

4 Выполнение лабораторной работы

```
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab06$ touch lab6-1.asm
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab06$ ls
lab6-1.asm  presentation  report
```

Создаём файл lab6-1.asm.

```
/home/petlin/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab06/lab6-1.asm
%include 'in_out.asm'
SECTION .bss
    buf1: RESB 80
SECTION .text
GLOBAL _start
_start:
mov eax,'6'
mov ebx,'4'
add eax,ebx
mov [buf1],eax
mov eax,buf1
call sprintLF
call quit
```

```
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab06$ nasm -f elf lab6-1.asm
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab06$ ld -m elf_i386 -o lab6-1 lab6-1.o
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab06$ ./lab6-1
j
```

Вводим в файл lab6-1.asm текст программы из листинга 6.1 и создаём исполняемый файл и запускаем его. Не получаем число 10.

```
lab6-1.asm          [----]  9 L:[  1+12  13/ 13] *(172 / 172b) <EOF>
%include 'in_out.asm'
SECTION .bss
    buf1: RESB 80
SECTION .text
GLOBAL _start
_start:
mov eax,6
mov ebx,4
add eax,ebx
mov [buf1],eax
mov eax,buf1
call sprintLF
call quit
```

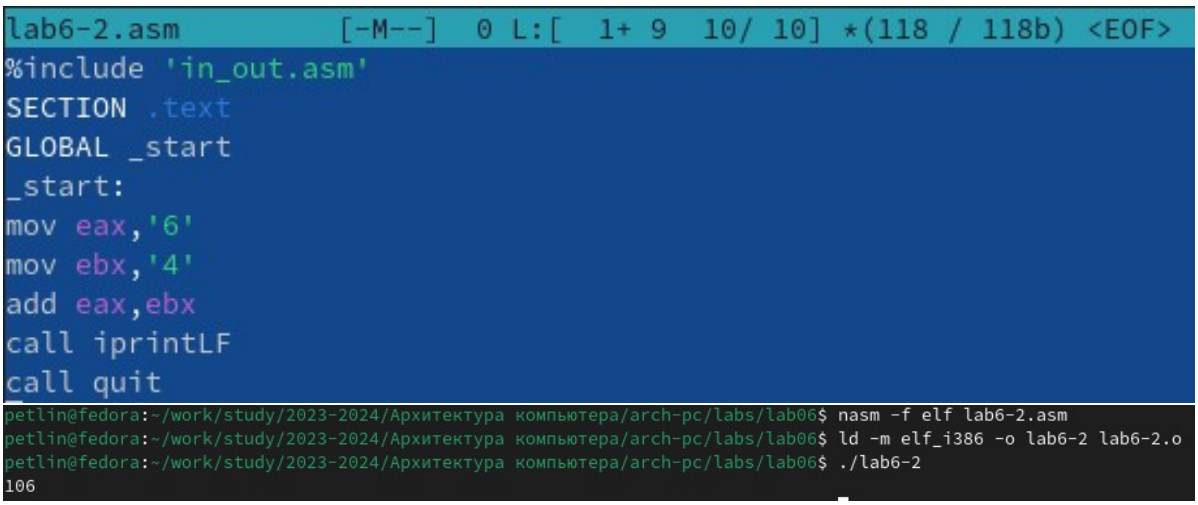
```
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab06$ nasm -f elf lab6-1.asm
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab06$ ld -m elf_i386 -o lab6-1 lab6-1.o
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab06$ ./lab6-1
```

ASCII Table

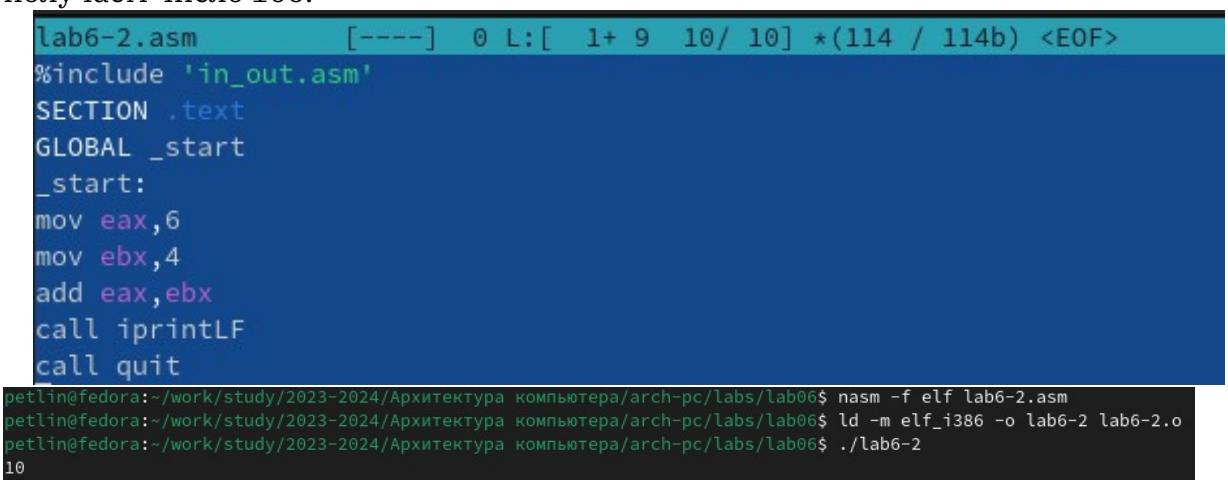
Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(	72	48	110	H	104	68	150	h
9	9	11		41	29	51	)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[	123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	|
29	1D	35		61	3D	75	=	93	5D	135	]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	

Далее изменяем текст программы и вместо символов, записывам в регистры числа. Создаём исполняемый файл и запускаем его. Снова не получаем число 10. Вывод содержит пустые символы (пробелы).

```
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab06$ touch lab6-2.asm
```



Создаём файл lab6-2.asm, вводим в него текст программы из листинга 6.2 и создаём исполняемый файл и запускаем его. В результате работы программы мы получаем число 106.



Аналогично предыдущему примеру изменим символы на числа. Создаём исполняемый файл и запускаем его. В результате работы программы мы получаем число 10.

```
lab6-2.asm          [BM--]  4 L:[  1+ 7   8/ 10] *(94  / 112b) 0032 0x020
%include 'in_out.asm'
SECTION .text
GLOBAL _start
_start:
mov eax,6
mov ebx,4
add eax,ebx
call iprint
call quit
```

```
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab06$ nasm -f elf lab6-2.asm
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab06$ ld -m elf_i386 -o lab6-2 lab6-2.o
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab06$ ./lab6-2
10petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab06$
```

Заменяем функцию iprintLF на iprint. Создаём исполняемый файл и запускаем его. В результате работы программы мы получаем число 10, однако в этот раз в конце программа не переходит на новую строку.

```
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab06$ touch lab6-3.asm
```

```
/home/petlin/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab06/lab6-3.asm
%include 'in_out.asm' ; подключение внешнего файла
SECTION .data
    div: DB 'Результат: ',0
    rem: DB 'Остаток от деления: ',0
SECTION .text
GLOBAL _start
_start:
    ; ---- Вычисление выражения
    mov eax,5 ; EAX=5
    mov ebx,2 ; EBX=2
    mul ebx ; EAX=EAX*EBX
    add eax,3 ; EAX=EAX+3
    xor edx,edx ; обнуляем EDX для корректной работы div
    mov ebx,3 ; EBX=3
    div ebx ; EAX=EAX/3, EDX=остаток от деления
    mov edi,eax ; запись результата вычисления в 'edi'
    ; ---- Вывод результата на экран
    mov eax,div ; вызов подпрограммы печати
    call sprint ; сообщения 'Результат: '
    mov eax,edi ; вызов подпрограммы печати значения
    call iprintLF ; из 'edi' в виде символов
    mov eax,rem ; вызов подпрограммы печати
    call sprint ; сообщения 'Остаток от деления: '
    mov eax,edx ; вызов подпрограммы печати значения
    call iprintLF ; из 'edx' (остаток) в виде символов
    call quit ; вызов подпрограммы завершения
```

```
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab06$ nasm -f elf lab6-3.asm
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab06$ ld -m elf_i386 -o lab6-3 lab6-3.o
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab06$ ./lab6-3
Результат: 4
Остаток от деления: 1
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab06$ mc
```

В качестве примера выполнения арифметических операций в NASM приведем программу вычисления арифметического выражения □(□) = (5 □ 2 + 3)/3. Создаём файл lab6-3.asm, вводим в него текст программы из листинга 6.3, создаём исполняемый файл и запускаем его. Результат работы программы:

```
Результат: 4
Остаток от деления: 1
```

```
lab6-3.asm          [----] 45 L:[  1+25  26/ 26] *(1320/1320b) <EOF>
%include 'in_out.asm' ; подключение внешнего файла
SECTION .data
    div: DB 'Результат: ',0
    rem: DB 'Остаток от деления: ',0
SECTION .text
GLOBAL _start
_start:
    ; ---- Вычисление выражения
    mov eax,4 ; EAX=5
    mov ebx,6 ; EBX=2
    mul ebx ; EAX=EAX*EBX
    add eax,2 ; EAX=EAX+3
    xor edx,edx ; обнуляем EDX для корректной работы div
    mov ebx,5 ; EBX=3
    div ebx ; EAX=EAX/3, EDX=остаток от деления
    mov edi,eax ; запись результата вычисления в 'edi'
    ; ---- Вывод результата на экран
    mov eax,div ; вызов подпрограммы печати
    call sprint ; сообщения 'Результат: '
    mov eax,edi ; вызов подпрограммы печати значения
    call iprintLF ; из 'edi' в виде символов
    mov eax,rem ; вызов подпрограммы печати
    call sprint ; сообщения 'Остаток от деления: '
    mov eax,edx ; вызов подпрограммы печати значения
    call iprintLF ; из 'edx' (остаток) в виде символов
    call quit ; вызов подпрограммы завершения
```

```
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab06$ nasm -f elf lab6-3.asm
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab06$ ld -m elf_i386 -o lab6-3 lab6-3.o
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab06$ ./lab6-3
Результат: 5
Остаток от деления: 1
```

Изменяем текст программы для вычисления выражения □(□) = (4 □ 6 + 2)/5. Создаём исполняемый файл и проверяем его работу. Вывод верен.

В качестве другого примера рассмотрим программу вычисления варианта задания по номеру студенческого билета, работающую по следующему алгоритму:

- вывести запрос на введение № студенческого билета
- вычислить номер варианта по формуле: (□□ mod 20) + 1, где □□ – номер студенческого билета (В данном случае □ mod □ – это остаток от деления □ на □).
- вывести на экран номер варианта.

```
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab06$ touch variant.asm
```

```
variant.asm         [-M--]  0 L:[  1+25  26/ 26] *(521 / 521b) <EOF>
%include 'in_out.asm'
SECTION .data
    msg: DB 'Введите № студенческого билета: ',0
    rem: DB 'Ваш вариант: ',0
SECTION .bss
    x: RESB 80
SECTION .text
GLOBAL _start
 _start:
 mov eax, msg
 call sprintLF
 mov ecx, x
 mov edx, 80
 call sread
 mov eax,x ; вызов подпрограммы преобразования
 call atoi ; ASCII кода в число, `eax=x`
 xor edx,edx
 mov ebx,20
 div ebx
 inc edx
 mov eax,rem
 call sprint
 mov eax,edx
 call iprintLF
 call quit
```

```
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab06$ nasm -f elf variant.asm
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab06$ ld -m elf_i386 -o variant variant.o
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab06$ ./variant
Введите № студенческого билета:
1132246846
Ваш вариант: 7
```

Создаём файл variant.asm, вводим в него текст программы из листинга 6.4, создаём исполняемый файл и запускаем его. Результат работы программы: 7. Однако, проверив результат работы программы, вычислив номер варианта аналитически, получаем 6. Изменим программу, чтобы результат был верен, для этого убираем строку inc edx, так как она добавляет 1 к нашему ответу:

```
variant.asm        [----]  0 L:[  1+24  25/ 25] *(512 / 512b) <EOF>
%include 'in_out.asm'
SECTION .data
    msg: DB 'Введите № студенческого билета: ',0
    rem: DB 'Ваш вариант: ',0
SECTION .bss
    x: RESB 80
SECTION .text
GLOBAL _start
 _start:
 mov eax, msg
 call sprintLF
 mov ecx, x
 mov edx, 80
 call sread
 mov eax,x ; вызов подпрограммы преобразования
 call atoi ; ASCII кода в число, `eax=x`
 xor edx,edx
 mov ebx,20
 div ebx
 mov eax,rem
 call sprint
 mov eax,edx
 call iprintLF
 call quit
```

```
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab06$ nasm -f elf variant.asm
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab06$ ld -m elf_i386 -o variant variant.o
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab06$ ./variant
Введите № студенческого билета:
1132246846
Ваш вариант: 6
```

Теперь программа работает правильно.

5 Ответы на вопросы

1. Какие строки листинга 6.4 отвечают за вывод на экран сообщения ‘Ваш вариант:’?
   Ответ:
   mov eax,rem
   call sprint

2. Для чего используется следующие инструкции?
   mov ecx, x
   mov edx, 80
   call sread
   Ответ: данные инструкции используются для чтения строки с введенными пользователем данными. Начальный адрес строки сохраняется в ecx, максимальное количетво символов которое может быть считано сохраняется в edx, a call sread производит чтение строки.
3. Для чего используется инструкция “call atoi”?
   Ответ: инструкция преобразовывает ASCII код в число
4. Какие строки листинга 6.4 отвечают за вычисления варианта?
   Ответ:
   mov eax,x ; вызов подпрограммы преобразования
   call atoi ; ASCII кода в число, eax=x
   xor edx,edx ; обнуление edx
   mov ebx,20 ; ebx=20

div ebx ; eax mod ebx = номер варианта

Eax сохраняет в себе номер стунденческого билета, ebx сохраняет себе делитель 20. Программа делит eax на ebx и остаток записывается в edx, поэтому он предварительно был обнулен (для корректной работы div). 5. В какой регистр записывается остаток от деления при выполнении инструкции “div ebx”?
Ответ: в регистр edx 6. Для чего используется инструкция “inc edx”?
Ответ: увеличивает значение в регистре edx на 1 7. Какие строки листинга 6.4 отвечают за вывод на экран результата вычислений?
Ответ:
mov eax,edx
call iprintLF
Происходит запись в регистр eax полученного значение остатка от деления, и с помощью инструкции call iprintLF производиться вывод на экран. (Необходимо записать остаток из регистра edx в регистр eax, так как инструкция выводит на экран значение из регистра eax).

6 Выполнение задания для самостоятельной работы

Напишем программу вычисления выражения предложенного в варианте, получившемся при выполнении лабораторной работы (6) □^3/2 + 1:

```
lab6-4.asm          [----]  0 L:[  1+30  31/ 31] *(592 / 592b) <EOF>
%include 'in_out.asm'
SECTION .data
    msg: DB 'Введите число x: ',0
    rem: DB 'Результат: ',0
SECTION .bss
    x: RESB 80
SECTION .text
GLOBAL _start
 _start:
 mov eax, msg
 call sprintLF ;вывели строку msg
 mov ecx, x
 mov edx, 80
 call sread ;считали ввод с клавиатуры
 mov eax,x
 call atoi ;eax=x
 mov ebx,eax ;ebx=x
 mul eax ;eax=x^2
 mul ebx ;eax=x^3
 xor edx,edx ;edx=0
 xor ebx,ebx ;ebx=0
 mov ebx,2 ;ebx=2
 div ebx ; eax/ebx=eax/2
 add eax,1 ;eax+1
 mov edi,eax
 mov eax,rem
 call sprint
 mov eax,edi
 call iprintLF
 call quit
```

```
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab06$ nasm -f elf lab6-4.asm
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab06$ ld -m elf_i386 -o lab6-4 lab6-4.o
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab06$ ./lab6-4
Введите число x:
2
Результат: 5
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab06$ ./lab6-4
Введите число x:
5
Результат: 63
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab06$ mc
```

Написав следующий код и создав исполняемый файл, проверяем его работу для предложенных x1=2, x2=5. Проверив результат работы программы аналитически, убеждаемся, что программа работает верно.

7 Выводы

Мы освоили арифметических инструкций языка ассемблера NASM.

Список литературы

1. GDB: The GNU Project Debugger. — URL: https://www.gnu.org/software/gdb/.

2. GNU Bash Manual. — 2016. — URL: https://www.gnu.org/software/bash/manual/.

3. Midnight Commander Development Center. — 2021. — URL: https://midnight-commander. org/.

4. NASM Assembly Language Tutorials. — 2021. — URL: https://asmtutor.com/.

5. Newham C. Learning the bash Shell: Unix Shell Programming. — O'Reilly Media, 2005. — 354 c. — (In a Nutshell). — ISBN 0596009658. — URL: http://www.amazon.com/Learningbash-Shell-Programming-Nutshell/dp/0596009658.

6. Robbins A. Bash Pocket Reference. — O'Reilly Media, 2016. — 156 c. — ISBN 978-1491941591.

7. The NASM documentation. — 2021. — URL: https://www.nasm.us/docs.php.

8. Zarrelli G. Mastering Bash. — Packt Publishing, 2017. — 502 c. — ISBN 9781784396879.

9. Колдаев В. Д., Лупин С. А. Архитектура ЭВМ. — М. : Форум, 2018.

10. Куляс О. Л., Никитин К. А. Курс программирования на ASSEMBLER. — М. : Солон-Пресс, 2017.

11. Новожилов О. П. Архитектура ЭВМ и систем. — М. : Юрайт, 2016.

12. Расширенный ассемблер: NASM. — 2021. — URL: https://www.opennet.ru/docs/RUS/nasm/.

13. Робачевский А., Немнюгин С., Стесик О. Операционная система UNIX. — 2-е изд. — БХВПетербург, 2010. — 656 с. — ISBN 978-5-94157-538-1.

14. Столяров А. Программирование на языке ассемблера NASM для ОС Unix. — 2-е изд. — М. : МАКС Пресс, 2011. — URL: http://www.stolyarov.info/books/asm_unix.

15. Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. — 874 с. — (Классика Computer Science).

16. Таненбаум Э., Бос Х. Современные операционные системы. — 4-е изд. — СПб. : Питер, 2015. — 1120 с. — (Классика Computer Science).