Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

Мегафакультет компьютерных технологий и управления

Факультет программной инженерии и компьютерной техники

Лабораторная работа №4
по основам профессиональной деятельности

Вариант: 34134

Группа: P3113

Студент: Андрей Григорьев

Преподаватель: ,,,,,,,,,,,,,,,,,,,,

г. Санкт-Петербург

Апрель, 2021

Задание:

Лабораторная работа №4

По выданному преподавателем варианту восстановить текст заданного варианта программы и подпрограммы (программного комплекса), определить предназначение и составить его описание, определить область представления и область допустимых значений исходных данных и результата, выполнить трассировку программного комплекса.

Введите номер варианта 34134

52F: + 0200	53D: EE0B	71D: AC01	72B: 00F6
530: EE18	53E: AE07	71E: F204	
531: AE16	53F: 0C00	71F: F003	
532: 0C00	540: D71D	720: 7E09	
533: D71D	541: 0800	721: F005	
534: 0800	542: 0700	722: F804	
535: 4E13	543: 4E05	723: 4C01	
536: EE12	544: EE04	724: 4C01	
537: AE0F	545: 0100	725: 6E05	
538: 0C00	546: ZZZZ	726: CE01	
539: D71D	547: YYYY	727: AE02	
53A: 0800	548: XXXX	728: EC01	
53B: 0740	549: FF0A	729: 0A00	
53C: 6E0C	----------	72A: 067B	

Выполнение работы:

Расшифровка текста исходной программы

РАССТАВЬ СВОИ IP просто по порядку

Адрес	Код	Мнемоника	Комментарии
52F	0200	CLA	Очистка регистра
	EE18	ST IP+18	Обнуляем переменную RES
	AE16	LD IP+16	Y --> AC, Загружаем в AC переменную Y
	0C00	PUSH	Кладём на вершину стека Y
	D71D	CALL 71D	Вызываем подпрограмму
	0800	POP	Кладём в аккумулятор результат работы подпрограммы
	0740	DEC	AC-1 --> AC
	4E13	ADD IP+13	AC+RES --> AC
	EE12	ST IP+12	AC --> RES
	AE0F	LD IP+F	X --> AC, Загружаем в AC переменную X
	0C00	PUSH	Кладём на вершину стека переменную X
	D71D	CALL 71D	Вызываем подпрограмму
	0800	POP	Кладём в аккумулятор результат работы подпрограммы
	0740	DEC	AC - 1 -> AC
	6E0C	SUB IP+C	AC-RES --> AC
	EE0B	ST IP+B	AC --> RES

	AE07	LD IP+7	Z --> AC, Загружаем в AC переменную Z
	0C00	PUSH	Кладём на вершину стека Z
	D71D	CALL 71D	Вызываем подпрограмму
	0800	POP	Кладём в аккумулятор результат работы подпрограммы
	0700	INC	AC+1 --> AC
	4E05	ADD IP+5	AC+RES --> AC
	EE04	ST IP+4	AC --> RES
	0100	HLT	Завершение работы программы
	ZZZ	Z	Переменная
	YYY	Y	Переменная
	XXX	X	Переменная
	FF0A	RES	Переменная, результат работы программы
-------	------	------------------	----------
71D	AC01	LD &1	Загрузка в AC первого сверху элемента стека.
	F204	BMI 04	Если элемент меньше или равен нуля, то переход на 0x723
	F003	BEQ 03	
	7E09	CMP IP+9	Сравниваем AC и COMPARATOR
	F005	BEQ 05	Если AC равен COMPARATOR, то переходим в 0x727

	F804	BEQ 04	Если AC меньше COMPARATOR, то переходим в 0x727
	4C01	ADD &1	Прибавляем к AC первый сверху элемент стека
	4C01	ADD &1	Прибавляем к AC первый сверху элемент стека
	6E05	SUB IP+5	Вычитаем из AC переменную ADDITIONAL
	CE01	JUMP IP+1	Пропускаем следующую операцию
	AE02	LD IP+2	Загружаем в AC переменную COMPARATOR
	EC01	ST &1	Загружаем содержимое AC в первый сверху элемент стека
	0A00	RET	Возвращаемся из подпрограммы
	067B	COMPARATOR	Переменная
	00F6	ADDITIONAL	Переменная

Описание программы

1) Расположение программы в памяти

ПОПРАВЬ СВОИ ЦИФРЫ

Переменные, поступающие на вход программы, расположены в ячейках

3B9 - 3BC

Сама программа расположена в ячейках 3A0 - 3B8

Вызываемая из основной программы подпрограмма расположена в ячейках 71F - 72C

Переменные для подпрограммы расположены в 72D , 72E

2) Область представления

X, Y, Z — обрабатываемые программой переменные, 16 разрядные знаковые числа

RES — результат работы программы, 16-разрядное знаковое число

COMPARATOR, ADDITIONAL - переменные, задающие работу подпрограммы, 16 разрядные знаковые числа

3) Назначение программы

Сначала определим семантику работы подпрограммы, записав алгоритм её работы на псевдокоде

```
let additional
let comparator
    func f(x):
        if (x ≤ 0 || x ≤ comparator):
            return comparator
        else:
            return 3x - additional
```

Теперь мы можем записать результат работы всей программы

```
    let res = 0
        func main(x,y,z):
            res = f(y) - 1
            res = f(x) - 1 - res
            res += f(z) + 1
            return res
```

4) Область допустимых значений

Рассмотрим два варианта

- $\text{COMPARATOR} \geq 0$

В таком случае, результатом функции f() всегда будет COMPARATOR (следует из логики работы подпрограммы).

Тогда, $X, Y, Z \in [-2^{15} + 1 ; 2^{15}]$, так как они не учавствуют не в каких арифметических операциях, кроме ± 1, и, как следствие, не могут вызвать переполнения при таком ОДЗ.

После второго вызова подпрограммы, в RES хранится f(z - 1) + f(x), что в нашем случае равно 2 * COMPARATOR. Чтобы не случилось переполнения, мы должны требовать: COMPARATOR $\in [0; 2^{14}]$.

Так как ADDITIONAL вообще не учавствует в исполнении программы при данных условиях, он может принимать любые значения.

ADDITIONAL $\in [-2^{15} ; 2^{15} - 1]$.

В переменной RES, после исполнения программы, будет лежать значение, равное: - COMPARATOR. Значит, RES $\in [-2^{14} ; 0]$.

■ COMPARATOR < 0

Если X,Y,Z < COMPARATOR, то подпрограмма возвращает COMPARATOR, и ограничения на такой случай мы уже рассматривали в первом пункте. Если же обрабатываемые переменные больше чем COMPARATOR и меньше нуля, то мы получаем:

$$\text{COMPARATOR} < X,Y,Z \leq 0$$

Затем подпрограмма вычисляет значение 5X + ADDITIONAL.

Чтобы не допустить переполнения, мы должны наложить ограничения на ADDITIONAL и COMPARATOR, так как именно эти две переменные задают возвращаемое из подпрограммы значение.

Чтобы 2 * (5X + ADDITIONAL) не вызвало переполнения из отрицательного числа в положительное:

COMPARATOR $\in [- 2^{9}; 0]$.

Чтобы 2 * (5X + ADDITIONAL) не вызвало переполнения в положительную сторону:

ADDITIONAL $\in [0; 2^{14} - 1]$.

Заметим, что на сами переменные X,Y,Z не накладывается дополнительных ограничений, так как все возможные переполнения с участием этих переменных уже учтены в ограничениях на COMPARATOR, ADDITIONAL:

X, Y, Z $\in [-2^{15} ; 2^{15} - 1]$.

Трассировка программы