[(10+20) \* (12-10)]

push 10

push 20

add

push 12

push 10

sub

mul

out

hlt

КОМАНДЫ (7 знаков)

1. push
2. pop
3. add
4. sub
5. mul
6. div
7. out
8. hlt
9. sin
10. cos
11. sqrt
12. in – считать число с клавиатуры и запихать в стек
13. dump
14. crack/hack
15. ; - комментарий

Нумеруем команды, т е как ассемблер. Получим аналог CPU

* Нужен ассемблер – переводит мнемонический код в «Байт-код»
* Также нужен дизассемблер
* После этого soft-processor исполняет «Байт-код»

**(Дробная арифметика – угореть и сделать double)**

* Исходный код на ассемблере
* Байт-код

**Разобраться с тем, как сделать ассемблер похожим на mingw!**

Храним программу как массив чисел – сначала как все double, потом неравномерно

Выбор формата

\*.jsm – файл с исходником

\*.jbc ­ - файл с программой в байт-коде

\*.jal – файл с листингом процесса ассемблирования

Листинг –

0x0000 | 1 10.5 | 0100000000000000 0204000F09FA6BC7 | push 10.5

0x0010 | 1 20.35 | 0100000000000000 04070BF000457C7H | push 20.35

0x0020 | 2……………………………………………………….

printf(“%016llx”); - вывести норм число

printf(“%A”); - вывод HEX дабла

rip – указатель на инструкцию

Регистры

Программа

Считаем дискриминант

**in**

**in**

**in**

**pop** rcx

**pop** rbx

**pop** rax

**push** rbx

**push** rbx

**mul**

**push** 4

**push** rax

**push** rax

**mul**

**mul**

**sub**

**hlt**

push/pop можно поделить на 5 команд, либо на 2 (у регистрового аргумент индекс – регистр), либо одна, но с доп аргументом – режим

Можно сделать вообще круто – и регистр + регистр, и регистр + конст, и конст + конст

В начале файла можно запихать сигнатуру

%n

Ужас:

#define DEF\_CMD(name, num, arg, code)\

CMD\_ ##name = arg,

enum CMD

{

#include “COMMANDS.H”

}

Domain-specific Language

Условные переходы

Ja

Jae

Jb

Jbe

Je

Jne

Jm (jump if Monday)

Задачи:

1) Квадратное уравнение

2) Рекурсивный алгоритм