Сборка примеров под ASM (80x86)

0. Общая часть

0.1 Приведены примеры для сборки
(п.1) ASM, 64 бит, Linux, консоль, линкер ld и gcc
(п.2) C (main) + ASM, 64 бит, Linux, консоль, линкер ld и gcc
(п.3) C (main) + ASM, 64 бит, Linux, **QtCreator**
0.2 Добавлено описание gdb (п. 4), включая графический интерфейс
0.3 Добавлено описание SASM IDE (п. 5)

1. ASM, 64 бит, Linux, консоль, линкер ld и gcc

1.1 Создаём файл mytest.asm

```
section .data
    a dw 185
section .text
global main
main:
    xor rax,rax
    mov ax,[a]
    not ax
    add ax,1
; выход
    mov eax,1
    mov ebx,0
    int 0x80
```

Примечания:
1) главную функцию называем main, а не _start для возможности использования линкера gcc
2) при использовании линкера ld явно используем системный вызов sys_exit (здесь через int 80h).
3) При использовании gcc можно указать ret

1.2 Трансляция
```
user@host:~$ nasm -g -f elf64 mytest.asm -o mytest.o -F dwarf
```

1.3 Линковка
1.3.1 Линковка с использованием ld
```
user@host:~$ ld -m elf_x86_64 -o mytest  mytest.o -z noexecstack -e
main
```

1.3.2 Линковка с использованием gcc
```
user@host:~$ gcc -o mytest mytest.o -m64 -no-pie -z noexecstack
```

1.3.3 Линковка с использованием ld при подключении методов из стандартной библиотеки libc
```
user@host:~$ ld -m elf_x86_64 -o mytest  mytest.o  -z noexecstack -e
main -lc -dynamic-linker /lib64/ld-linux-x86-64.so.2
```

1.4 Команды можно собрать в Makefile, к примеру

```
all:
	nasm -g -f elf64 mytest.asm -o mytest.o -F dwarf
	ld -m elf_x86_64 -o mytest  mytest.o -z noexecstack -e main
```

Запуск сборки командой **make**

```
user@host:~$ make
```

2 C (main) + ASM, 64 бит, Linux, консоль, линкер ld и gcc

2.1 Создание файлов

2.1.1 Создать ctest.c

```
#include <stdlib.h>
#include <stdio.h>
extern void asm_func(void);
int main(void)
{
    asm_func();
    printf("hello!\n");
    return 0;
};
```

2.1.2 Создать atest.asm

```
section .data
    a dw 100
section .text
global asm_func
asm_func:
    mov ax, [a]
    inc ax
    ret
```

2.2 Создание объектных файлов

2.2.1 Трансляция asm файла

```
user@host:~$ nasm -g -f elf64 atest.asm -o atest.o -F dwarf
```

2.2.2 Компиляция C файла

```
user@host:~$ gcc -c -g -ggdb -o ctest.o ctest.c
```

2.3 Линковка

2.3.1 Линковка с использованием ld

```
user@host:~$ ld -m elf_x86_64 -o mytest atest.o ctest.o -z noexecstack
```

2.3.2 Линковка с использованием gcc

```
user@host:~$ gcc -o mytest atest.o ctest.o -m64 -no-pie -z noexecstack
```

2.3.3 Линковка с использованием ld при подключении методов из стандартной библиотеки libc

```
user@host:~$ ld -m elf_x86_64 -o mytest atest.o ctest.o -z
noexecstack -lc  dynamic-linker /lib64/ld-linux-x86-64.so.2
```

2.4 Команды можно собрать в Makefile, к примеру

```
all:
```

```
gcc -c -g -ggdb -o ctest.o ctest.c
nasm -g -f elf64 atest.asm -o atest.o -F dwarf
gcc -o mytest atest.o ctest.o -m64 -no-pie -z noexecstack
```

Запуск сборки командой **make**

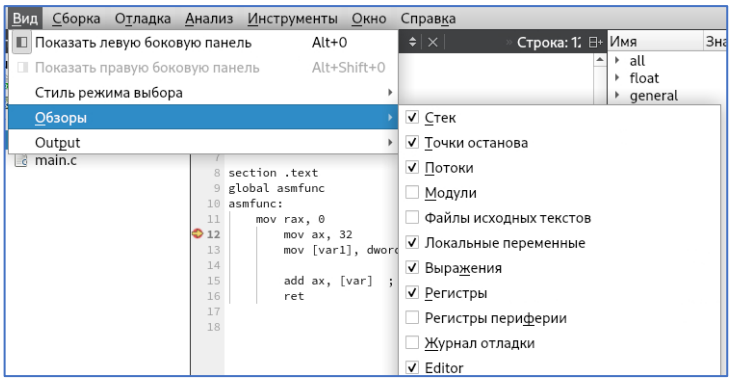
```
user@host:~$ make
```

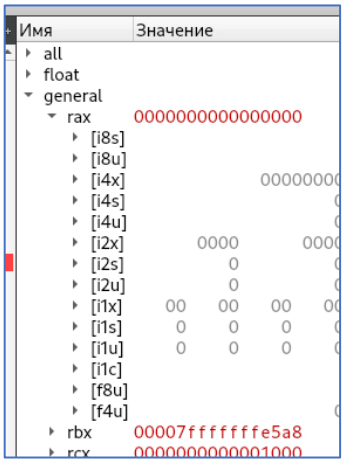
3 C (main) + ASM, 64 бит, Linux, QtCreator

3.1 Создаем проект, приводим pro-файл к виду

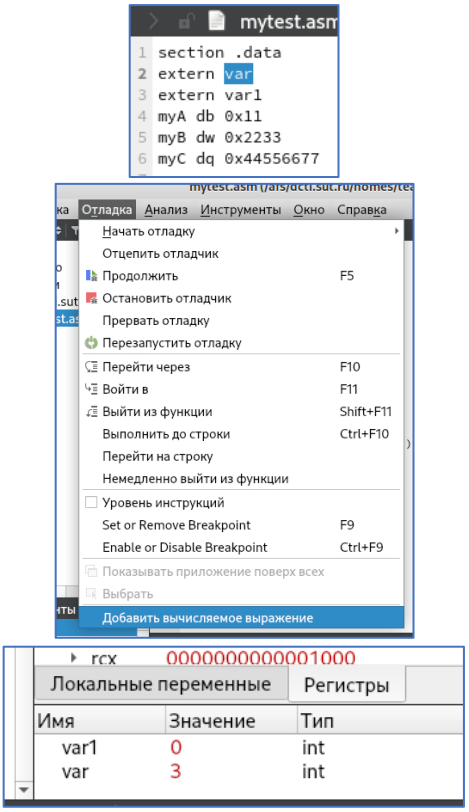
```
TEMPLATE = app
CONFIG += console
CONFIG -= app_bundle
CONFIG -= qt

QMAKE_EXTRA_COMPILERS += nasm
QMAKE_LFLAGS += -no-pie -z noexecstack
NASM_EXTRAFLAGS = -f elf64 -g -F dwarf
OTHER_FILES += $$NASM_SOURCES
nasm.output = ${QMAKE_FILE_BASE}.o
nasm.commands = nasm $$NASM_EXTRAFLAGS -o ${QMAKE_FILE_BASE}.o
${QMAKE_FILE_NAME}
nasm.input = NASM_SOURCES
SOURCES += \
    main.c

NASM_SOURCES += \
    mytest.asm
```

Все C-файлы указываем в SOURCES, все asm-файлы – в NASM_SOURCES
Сборка и запуск – штатными средствами QtCreator.
Можно выполнять отладку. После запуска отладки для отображения регистров выбрать **Вид->Обзоры->Регистры**





Для отображения значений глобальных переменных, выделить нужную переменную и нажать использовать «Отладка->Добавить вычисляемое выражение».



Локальные переменные отображаются автоматически при входе в функцию.

4 Отладка в gdb в консоли

4.4.1 запуск отладчика
Запуск сборки командой **make**

```
user@host:~$ gdb mytest
```

4.4.2 назначение точки останова

```
(gdb) b main
```

4.4.3 старт исполнения или продолжение до точки останова

```
(gdb) r        //старт
(gdb) c        //продолжение
```

4.4.4 продолжение исполнения (один шаг)
заход внутрь функции

```
(gdb) s
```

без захода внуть функции

```
(gdb) n
```

4.4.5 печать регистра

```
(gdb) print /format $reg
```

где
reg – имя регистра
format – формат представления, есть отсутствует, то по умолчанию **d**
Возможные значения:
x – шестнадцатеричное
d – десятичное знаковое
u – десятичное беззнаковое
o – восьмеричное значение
t – двоичное
a – адрес
c – символьное
f – плавающая точка
s – строковое
z – шестнадцатеричное с добавлением лидирующих нулей
Пример

```
(gdb) print /x $rax
```

4.4.6 печать регистра при каждом шаге отладки

```
(gdb) display $ax
```

или

```
(gdb) display /x $ax
```

4.4.7 печать всех регистров

```
(gdb) info registers
```

4.4.8 печать регистров сопроцессора

```
(gdb) info float
```

4.4.9 печать переменной

```
(gdb) print a
```

или

```
(gdb) print /x a
```

или

```
(gdb) print (word) a        //прим: если просит, указать тип явно
```

4.4.10 инспекция памяти с адреса **addr**

```
(gdb) x /format &addr
```

где
addr – метка
/format – формат вывода вида /NFU
N – число повторений, по умолчанию 1
F – формат отображения, смотри про print (п 4.4.5)
U – размер единицы отображения
b - байт

```
      h – два байта
      w – четыре байта
      g – восемь байтов
Пример
      (gdb) x /4xg &var1
Отобразит 4 раза (4) по 8 байт (g) в шестнадцатеричном формате (x),
начиная с адреса метки var1
```

4.4.11 выход

```
      (gdb) q
```

4.4.12 автоматизация выполнения команд.

Создаем текстовый фай, к примеру `gdbcomm`, в него добавляем команды, которые надо выполнить при начале отладки, к примеру:

```
b asmfunc
display $eax
r
```

Запускаем отладку, указав доп.параметр -x

```
user@host:~$ gdb mytest -x gdbcomm
```

4.4.13 Графический интерфейс gdb

Запускаем с опцией `-tui`

```
user@host:~$ gdb mytest -tui
```

для отображения регистров вводим команду

```
(gdb) layout regs
```

если отображение `asm` идет в формате ATT, то для отображения в формате `intel` вводим команду

```
(gdb) set disassembly-flavor intel
```

5. SASM IDE

5.1 Брать здесь:

https://download.opensuse.org/repositories/home:/Dman95/

Проверял на Windows 10 и Astra Linux SE 1.6 (клон Debian)

5.2 Настройка примитивна

По умолчанию результат выкладывает в `/tmp/sasm`

Во встроенной инструкции сказано, как поменять параметры сборки, чтоб сохранялось в другое место и с другим именем.

5.3 Примеры программ

```
в /usr/share/sasm/projects/
или C:\Program Files (x86)\SASM\Projects
```

Для nasm это `NASMHello.asm` и `NASMHello64.asm`. В зависимости от файла выбрать в настройке режим сборки x86 или x64.

5.4 Отображение регистров включается отдельно (Ctrl-R).