Сборка примеров под ASM (80x86)

0. Общая часть

0.1 Приведены примеры для сборки
- (п.1) чистый ASM 16 бит DOS, консольная сборка
- (п.2) ASM + C, 16 бит DOS, консольная сборка
- (п.3) ASM + C, 16 бит DOS, через IDE
- (п.4) чистый ASM 32 бит Linux, консольная сборка
- (п.5) ASM + C, 32 бит Linux, консольная сборка
- (п.6) чистый ASM 64 бит Linux, консольная сборка
- (п.7) ASM + C, 64 бит Linux, консольная сборка

0.2 Версии среды разработки под DOS, установленные на учебных машинах и машине преподавателя, различаются! На учебной машине работает сборка в IDE, но дебагер не отображает исходный текст.
0.3 Запуск DOS выполняется через «Приложения/Системные/DOS эмулятор».
0.4 Домашняя директория отображается в dosemu как диск D:.
0.5 Среда разработки под DOS развернута на F:\borlandc на учебной машине.
0.5 Для сборки под 32 бит требуется установка пакета libc-dev-i386
0.6 Добавлено описание SASM IDE

1. Чистый ASM 16 бит DOS, консольная сборка

1.1 Создаем файл mytest.asm

```
.model large
.stack 100h
.data
message db 'hello', 13, 10, '$'
var1    dw 0aa55h
.code
start:
   mov ax, @data
   mov ds, ax
   mov ax, var1
   inc ax
   mov var1, ax
   mov ah, 9
   mov dx, OFFSET message
   int 21h
   mov ah, 4ch;
   int 21h
end
```

1.2 Сборка
```
tasm /zi mytest.asm
```

1.3 Линковка
```
tlink /v mytest.obj
```

1.4 Отладка
td mytest.exe
Примечание: td 3.1 не отображает исходники, но работать можно. Версия td 3.2 работает корректно.

1.5 Интерфейс
F8 – шаг
Alt+F5 – консоль
Alt+V - C – просмотр регистров

1.6 Выход
Alt+X – выход
Ctrl+F10 – отвязать курсор мыши от DosBox

2. ASM + C 16 бит DOS, консольная сборка

2.0 Примечание: отработало только для модели памяти «small», увы. Надо разбираться с опциями сборки

2.1 Создание файлов
2.1.1 Создаем ctest.c
```
#include <stdlib.h>
#include <stdio.h>
extern void ASM_FUNC(void);   //имя функции большими буквами
int main(void)
{
    int a = 0;
    ASM_FUNC();               //имя функции большими буквами
    printf("hello!\n");
    a ++;
    return 0;
};
```

2.1.2 Создаем atest.asm
```
.model small, C
.stack 100h
.data
    ;extrn aaa:word
    var1 dw 0aa55h
    Message db 'myProgMessage', 13, 10, '$'
.code
public asm_func
asm_func proc C
    push ax
    mov ax, @data
    mov ds, ax
    mov ax, OFFSET var1
    mov ax, var1
    inc ax
    mov var1, ax
    pop ax
    ret
asm_func endp
end
```

2.1.3 Создаем build.bat

```
bcc -c -M -v -If:\borlandc\include ctest.c
tasm /zi atest.asm
bcc /v -L f:\borlandc\lib ctest.obj atest.obj
```

2.2 Сборка

build.bat

2.3 Отладка

```
td ctest.exe
```

Примечание: td 3.1 не отображает исходники, но работать можно. Версия td 3.2 работает корректно.

2.4 Интерфейс

Смотри п. 1.5 (Интерфейс)

3. ASM + C 16 бит DOS, IDE

3.1 Создаем проект с именем mytest.prj

bc mytest.prj

3.2 Создаем файлы в IDE (F10-File-New-Save As)

3.2.1 Создаем ctest.c

```
#include <stdlib.h>
#include <stdio.h>
extern void asm_func(void);   //имя функции маленькими буквами
int main(void)
{
    int a = 0;
    asm_func();               //имя функции маленькими буквами
    printf("hello!\n");
    a ++;
    return 0;
};
```

3.2.2 Создаем atest.asm

```
.model large, C
.data
    var1 dw 0aa55h
.code
public asm_func
asm_func proc C
    mov ax, var1
    inc ax
    mov var1, ax
    ret
asm_func endp
end
```

3.3 Добавляем файлы ctest.c и atest.asm в проект (Alt+W – P, далее Insert и выбрать файл)

3.4 Проверить модель памяти и установить как в ASM модуле (Alt+O-C-C)

3.5 Сборка в IDE
Alt+C – M или F9

3.6 Запуск
F10 – R или Alt+R – R или Ctrl+F9

3.7 Отладка Alt+R и далее G(go to cursor) или T(trace into), или S(step over)
или F4(go to cursor), или F7(trace into), или F8(step over).
Примечание: отладка без возможности просмотра asm кода

3.8 Выход Alt+X

3.9 Сборка в консоли
bc /b mytest.prj

3.10 Запуск в отладчике
td mytest.exe

4. Чистый ASM 32 бит Linux, консольная сборка

4.1 Создаём файл mytest.asm

```
section .data
    a dw 185
section .text
global _start
_start:
main:           ; данная метка добавлена для отладки
    xor eax,eax
    mov ax,[a]
    not ax
    add ax,1
fin:            ; данная метка добавлена для отладки
    mov eax,1
    mov ebx,0
    int 0x80
```

4.2 Сборка
nasm -g -f elf mytest.asm -l mytest.lst -F dwarf

4.3 Линковка
ld -m elf_i386 -o mytest mytest.o

4.4 Отладка

4.4.1 запуск отладчика
gdb mytest

4.4.2 назначение точки останова
(gdb) b main

4.4.3 старт исполнения или продолжение до точки останова

```
(gdb) r
```

4.4.4 продолжение исполнения (один шаг)

```
(gdb) s
```

или

```
(gdb) ni
```

4.4.5 печать регистра

```
(gdb) print $ax
```

или

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

4.4.6 печать всех регистров

```
(gdb) info registers
```

4.4.6 печать регистров сопроцессора

```
(gdb) info float
```

4.4.7 печать переменной

```
(gdb) print a
```

или

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

или

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

4.4.8 выход

```
(gdb) q
```

5. ASM + C, 32 бит Linux, консольная сборка

5.1 Создание файлов

5.1.1 Создать ctest.c

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

5.1.2 Создать atest.asm

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

5.1.3 Создать makefile

```
all:
          gcc -m32 -g -ggdb -c -o ctest.o ctest.c
          nasm -g -f elf atest.asm -l atest.lst -F dwarf
          gcc -m32 -o ctest ctest.o atest.o -fno-pie -no-pie
```

5.2 Сборка

```
          make
```

5.3 Отладка

Смотри 4.4 (Отладка)

6. чистый ASM 64 бит Linux, консольная сборка

6.1 Создаем файл mytest.asm

```
section .data
    a dw 185
section .text
global _start
_start:
main:           ; данная метка добавлена для отладки
    xor rax,rax   ; отличие от 32 примера только в это строчке
    mov ax,[a]
    not ax
    add ax,1
fin:            ; данная метка добавлена для отладки
    mov eax,1
    mov ebx,0
    int 0x80
```

6.2 Сборка

```
nasm -g -f elf64 mytest.asm -l mytest.lst -F dwarf
```

6.3 Линковка

```
ld -o mytest mytest.o
```

6.4 Отладка

Смотри 4.4 (Отладка)

7. ASM + C, 64 бит Linux, консольная сборка

7.1 Создание файлов

7.1.1 Создать ctest.c

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

7.1.2 Создать atest.asm

```
section .data
    a dw 100
section .text
global asm_func
asm_func:
    push rax          ; отличие от 32 бит
    mov ax, [a]
    inc ax
    mov [a], ax
    pop rax           ; отличие от 32 бит
    ret
```

7.1.3 Создать makefile

```
all:
          gcc -g -ggdb -c -o ctest.o ctest.c
          nasm -g -f elf64 atest.asm -l atest.lst -F dwarf
          gcc -o ctest ctest.o atest.o -fno-pie -no-pie
```

7.2 Сборка

```
          make
```

7.3 Отладка

Смотри 4.4 (Отладка)

8. SASM IDE

8.1 Брать здесь:

https://download.opensuse.org/repositories/home:/Dman95/
Проверял на Windows 10 и Astra Linux SE 1.6 (клон Debian)

8.2 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.

8.3 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.

8.4 Отображение регистров включается отдельно (Ctrl-R). Регистры сопроцессора