

Ассемблер x86-64 (AMD64). Инструменты

The image displays the SASM (Simple Assembler) interface, which includes a code editor, a disassembler, and a debugger. The main window shows the assembly code for a program named 'jump.asm'. The code defines constants, declares variables, and implements a 'main' function that compares two numbers and prints the result.

```
1 ; jump.asm
2 extern printf
3 section .data
4     bNum db 123
5     wNum dw 12345
6     dNum dd 1234567890
7     qNum1 dq 1234567890123456789
8     qNum2 dq 123456
9     qNum3 dq 3.14
10 section .bss
11 section .text
12 global main
13 main:
14     push rbp
15     mov rbp, rsp
16     mov rax, -1 ; fill rax with 1s
17     mov al, byte [bNum] ; does NOT clear upper bits of rax
18     xor rax, rax ; clear rax
19     mov al, byte [bNum] ; now rax contains the correct value
20
21     mov rax, -1 ; fill rax with 1s
22 ) disassemble main
23 of assembler code for function main:
24 000000000401110 <->: push %rbp
25 000000000401111 <->: mov %rsp,%rbp
26 000000000401114 <->: mov $0xffffffffffffffff,%rax
27 00000000040111b <->: mov 0x404028,%al
28 000000000401122 <->: xor %rax,%rax
29 000000000401125 <->: mov 0x404028,%al
30 00000000040112c <->: mov $0xffffffffffffffff,%rax
31 000000000401133 <->: mov 0x404029,%ax
32 00000000040113e <->: xor %rax,%rax
33 000000000401146 <->: mov 0x404029,%ax
34 00000000040114e <->: mov $0xffffffffffffffff,%rax
35 000000000401154 <->: mov 0x40402b,%eax
36 00000000040115b <->: mov $0xffffffffffffffff,%rax
37 000000000401163 <->: mov 0x40402f,%rax
38 00000000040116b <->: mov %rax,0x404037
39 000000000401170 <->: mov $0x1234,%eax
40 000000000401176 <->: movq 0x40403f,%xmm0
41 000000000401179 <->: mov %rbp,%rsp
42 00000000040117c <->: pop %rbp
43 00000000040117e <->: ret
44 00000000040117e <->: xchg %ax,%ax
45 ) of assembler dump.
```

The debugger window shows the execution of the program. The registers window displays the state of the CPU registers, and the backtrace window shows the current execution point. The output window shows the program's output: "Hello World!".

Registers:

Register	Value
eax	0x0
ecx	0x0040904
edx	0xd
ebx	0x1
esp	0xffffd940
ebp	0x0
esi	0x0
edi	0x0
ebp	0x0040808f
eflags	0x202
cs	0x23
ss	0x2b
ds	0x2b
es	0x2b
fs	0x0
gs	0x0

Backtrace:

```
0) 0x0040808f in _start ()
```

Breakpoint 2, 0x0040808f in _start ()

Используемые источники

Сайт программы: <https://nasm.us/>

Википедия: Ассемблер - <https://ru.wikipedia.org/wiki/Ассемблер>

Assembler Linux: <https://habr.com/ru/sandbox/26864/>

Команды x86, x86-64: <http://ccfit.nsu.ru/~kireev/lab2/lab2com.htm>

Регистры x86, x86-64: <http://ccfit.nsu.ru/~kireev/lab2/lab2reg.htm>

Онлайн компиляторы, использующие NASM x86-64:

https://www.mycompiler.io/new/asm-x86_64

<https://ideone.com/>

Информация об архитектуре и Ассемблере процессоров Intel. Раздел про NASM:
<http://softcraft.ru/edu/comparch/ref/asm86/>

Hello, world. Компиляция в нотацию AT&T. GNU Assembler (gas)

```
#include <stdio.h>
```

```
int main() {  
    printf("Hello, world!\n");  
}
```

```
$ gcc -S hello.c
```

```
» .file» "hello.c"  
» .text  
» .section» .rodata  
.LC0:  
» .string» "Hello, world!"  
» .text  
» .globl» main  
» .type» main, @function  
main:  
.LFB0:  
» .cfi_startproc  
» pushq» %rbp  
» .cfi_def_cfa_offset 16  
» .cfi_offset 6, -16  
» movq» %rsp, %rbp  
» .cfi_def_cfa_register 6  
» leaq» .LC0(%rip), %rax  
» movq» %rax, %rdi  
» call» puts@PLT  
» movl» $0, %eax  
» popq» %rbp  
» .cfi_def_cfa 7, 8  
» ret  
» .cfi_endproc  
.LFE0:  
» .size» main, .-main  
» .ident» "GCC: (GNU) 11.1.0"  
» .section» .note.GNU-stack,"",@progbits
```

Hello, world. Компиляция в нотацию AT&T. GNU Assembler (gas)

```
#include <stdio.h>
```

```
int main() {  
    printf("Hello, world!\n");  
}
```

```
$ gcc -S -masm=intel hello.c
```

```
» .file» "hello.c"  
» .intel_syntax noprefix  
» .text  
» .section» .rodata  
.LC0:  
» .string» "Hell0, world!"  
» .text  
» .globl» main  
» .type» main, @function  
main:  
.LFB0:  
» .cfi_startproc  
» push» rbp  
» .cfi_def_cfa_offset 16  
» .cfi_offset 6, -16  
» mov» rbp, rsp  
» .cfi_def_cfa_register 6  
» lea» rax, .LC0[rip]  
» mov» rdi, rax  
» call» puts@PLT  
» mov» eax, 0  
» pop» rbp  
» .cfi_def_cfa 7, 8  
» ret  
» .cfi_endproc  
.LFE0:  
» .size» main, .-main  
» .ident» "GCC: (GNU) 11.1.0"  
» .section» .note.GNU-stack,"",@progbits
```

Hello, world. Сравнение нотаций. GNU Assembler (gas)

```
$ gcc -S hello.c
```

```
» .file» "hello.c"
» .text
» .section» .rodata
.LC0:
» .string» "Hello, world!"
» .text
» .globl» main
» .type» main, @function
main:
.LFB0:
» .cfi_startproc
» pushq» %rbp
» .cfi_def_cfa_offset 16
» .cfi_offset 6, -16
» movq» %rsp, %rbp
» .cfi_def_cfa_register 6
» leaq» .LC0(%rip), %rax
» movq» %rax, %rdi
» call» puts@PLT
» movl» $0, %eax
» popq» %rbp
» .cfi_def_cfa 7, 8
» ret
» .cfi_endproc
.LFE0:
» .size» main, .-main
» .ident» "GCC: (GNU) 11.1.0"
» .section» .note.GNU-stack,"",@progbits
```

```
$ gcc hello.s
```

```
$ gcc -S -masm=intel hello.c
```

```
» .file» "hello.c"
» .intel_syntax noprefix
» .text
» .section» .rodata
.LC0:
» .string» "Hello, world!"
» .text
» .globl» main
» .type» main, @function
main:
.LFB0:
» .cfi_startproc
» push» rbp
» .cfi_def_cfa_offset 16
» .cfi_offset 6, -16
» mov» rbp, rsp
» .cfi_def_cfa_register 6
» lea» rax, .LC0[rip]
» mov» rdi, rax
» call» puts@PLT
» mov» eax, 0
» pop» rbp
» .cfi_def_cfa 7, 8
» ret
» .cfi_endproc
.LFE0:
» .size» main, .-main
» .ident» "GCC: (GNU) 11.1.0"
» .section» .note.GNU-stack,"",@progbits
```

Hello, world. Использование Netwide Assembler (NASM)

```
; hello.asm
section .data
    msg     db          "Hello, world!",10,0
section .bss
section .text
    global main
main:

    mov    rax, 1        ; 1 = write
    mov    rdi, 1        ; 1 = to stdout
    mov    rsi, msg      ; string to display in rsi
    mov    rdx, 14       ; length of the string, without 0
    syscall                ; display the string
    mov    rax, 60       ; 60 = exit
    mov    rdi, 0        ; 0 = success exit code
    syscall                ; quit
```

NASM. Структура программы

section .data

Объявление и определение инициализируемых данных. Эквивалент глобальных переменных и констант. Выделяется память при ассемблировании и компоновки. Имя переменной обозначает начальный адрес переменной в памяти.

Форматы:

`<variable name> <type> <value>`
`<constant name> equ <value>`

	Тип	Длина	Название
db		8 бит	Байт
dw		16 бит	Слово
dd		32 бита	Двойное слово
dq		64 бита	Учетверенное (двойное длинное) слово

NASM. Структура программы

section .bss

Block Started by Symbol. В раздел помещаются неинициализированные переменные. Переменные не содержат начальных значений. Память для них резервируется во время выполнения.

Формат:

<variable name> <type> <number>

Тип	Длина	Название
resb	8 бит	Байт
resw	16 бит	Слово
resd	32 бита	Двойное слово
resq	64 бита	Учетверенное (двойное длинное) слово

NASM. Структура программы

section .text

Содержит код программы.
Начинается с инструкций:

global main
main:

```
; hello.asm
section .data
    msg      db      "Hello, world!",10,0
section .bss
section .text
    global main
main:

    mov     rax, 1      ; 1 = write
    mov     rdi, 1      ; 1 = to stdout
    mov     rsi, msg    ; string to display in rsi
    mov     rdx, 14     ; length of the string, without 0
    syscall          ; display the string
    mov     rax, 60     ; 60 = exit
    mov     rdi, 0      ; 0 = success exit code
    syscall          ; quit
```

Hello, world. Компиляция и сборка

```
#makefile for hello.asm
```

```
hello: hello.o
```

```
» gcc -o hello hello.o -no-pie
```

```
hello.o: hello.asm
```

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

-f elf64 - Executable and Linkable Format for 64-bit

-g - необходимо включить отладочную информацию в специальном формате отладки, определенном после ключа -F

-F dwarf - отладочный формат dwarf (Debug With Attributed Record Format)

-l hello.lst - генерации файла листинга .lst

-no-pie - не создавать перемещаемый код

Hello, world. Листинг программы

```
1          ; hello.asm
2          section .data
3 00000000 48656C6C6F2C20776F-      msg      db      "Hello, world!",10,0
3 00000009 726C64210A00
4
4          section .bss
5          section .text
6          global main
7          main:
8
9 00000000 B801000000      mov     rax, 1      ; 1 = write
10 00000005 BF01000000      mov     rdi, 1      ; 1 = to stdout
11 0000000A 48BE-          mov     rsi, msg    ; string to display in rsi
11 0000000C [00000000000000000000000000000000].
12 00000014 BA0E000000      mov     rdx, 14     ; length of the string, without 0
13 00000019 0F05          syscall            ; display the string
14 0000001B B83C000000      mov     rax, 60     ; 60 = exit
15 00000020 BF00000000      mov     rdi, 0      ; 0 = success exit code
16 00000025 0F05          syscall            ; quit
17
```

Инструменты для работы. Отладчик GDB

Консольный отладчик, применяемый практически для отладки приложений, написанных на различных языках. Часто используется в качестве фонового в различных графических средствах отладки.

Википедия: Отладчик GDB. https://ru.wikipedia.org/wiki/GNU_Debugger

Полное руководство с использованием отладчика GDB:
<https://russianblogs.com/article/2438428885/>

Краткий гайд по использованию GDB: <https://habr.com/ru/post/491534/>

Онлайн отладчик: <https://www.onlinegdb.com/>

Как пользоваться gdb: <https://losst.ru/kak-polzovatsya-gdb>

Инструменты для работы. Отладчик GDB

Основные команды отладчика

`$ gdb <program>` – *запуск отладчика с загрузкой отлаживаемой программы*

`(gdb) run` – *выполнение загруженной программы*

`(gdb) list` – *печать первых 10 строк программы (list или Enter) – следующие 10 строк*

`(gdb) list n,m` – *печать со строки с номером n по строку с номером m*

`(gdb) disassemble main` – *дизассемблирование*

`(gdb) set disassembly-flavor intel` – *установка дизассемблирования в стиле intel*

`.gdbinit` – *конфигурационный файл в домашнем каталоге*

Инструменты для работы. Отладчик GDB

```
(gdb) disassemble main
Dump of assembler code for function main:
   0x0000000000401110 <+0>:      mov     $0x1,%eax
=>  0x0000000000401115 <+5>:      mov     $0x1,%edi
   0x000000000040111a <+10>:     movabs  $0x404028,%rsi
   0x0000000000401124 <+20>:     mov     $0xe,%edx
   0x0000000000401129 <+25>:     syscall
   0x000000000040112b <+27>:     mov     $0x3c,%eax
   0x0000000000401130 <+32>:     mov     $0x0,%edi
   0x0000000000401135 <+37>:     syscall
   0x0000000000401137 <+39>:     nopw   0x0(%rax,%rax,1)
End of assembler dump.
```

```
(gdb) set disassembly-flavor intel
(gdb) disassemble main
Dump of assembler code for function main:
   0x0000000000401110 <+0>:      mov     eax,0x1
=>  0x0000000000401115 <+5>:      mov     edi,0x1
   0x000000000040111a <+10>:     movabs  rsi,0x404028
   0x0000000000401124 <+20>:     mov     edx,0xe
   0x0000000000401129 <+25>:     syscall
   0x000000000040112b <+27>:     mov     eax,0x3c
   0x0000000000401130 <+32>:     mov     edi,0x0
   0x0000000000401135 <+37>:     syscall
   0x0000000000401137 <+39>:     nop     WORD PTR [rax+rax*1+0x0]
End of assembler dump.
```

Инструменты для работы. Отладчик GDB

```
(gdb) list 1,18
1      ; hello.asm
2      section .data
3          msg      db      "Hello, world!",10,0
4      section .bss
5      section .text
6          global main
7      main:
8
9          mov     rax, 1      ; 1 = write
10         mov     rdi, 1      ; 1 = to stdout
11         mov     rsi, msg    ; string to display in rsi
12         mov     rdx, 14    ; length of the string, without 0
13         syscall           ; display the string
14         mov     rax, 60    ; 60 = exit
15         mov     rdi, 0      ; 0 = success exit code
16         syscall           ; quit
17
```

Инструменты для работы. Отладчик GDB

```
(gdb) set disassembly-flavor intel
(gdb) disassemble main
Dump of assembler code for function main:
   0x0000000000401110 <+0>:      mov     eax,0x1
=>  0x0000000000401115 <+5>:      mov     edi,0x1
   0x000000000040111a <+10>:     movabs rsi,0x404028
   0x0000000000401124 <+20>:     mov     edx,0xe
   0x0000000000401129 <+25>:     syscall
   0x000000000040112b <+27>:     mov     eax,0x3c
   0x0000000000401130 <+32>:     mov     edi,0x0
   0x0000000000401135 <+37>:     syscall
   0x0000000000401137 <+39>:     nop     WORD PTR [rax+rax*1+0x0]
End of assembler dump.
```

```
(gdb) x/s 0x404028
0x404028 <msg>: "Hello, world!\n"
```

```
(gdb) x/c 0x404028
0x404028 <msg>: 72 'H'
```

```
(gdb) x/s &msg
0x404028 <msg>: "Hello, world!\n"
(gdb) □
```

```
(gdb) x/2x 0x401110
0x401110 <main>:      0xb8      0x01
```

```
(gdb) x/14c 0x404028
0x404028 <msg>: 72 'H' 101 'e' 108 'l' 108 'l' 111 'o' 44 ',' 32 ' ' 119 'w'
0x404030:      111 'o' 114 'r' 108 'l' 100 'd' 33 '!' 10 '\n'
(gdb) x/14d 0x404028
0x404028 <msg>: 72      101      108      108      111      44      32      119
0x404030:      111      114      108      100      33      10
(gdb) x/14x 0x404028
0x404028 <msg>: 0x48      0x65      0x6c      0x6c      0x6f      0x2c      0x20      0x77
0x404030:      0x6f      0x72      0x6c      0x64      0x21      0x0a
```


Инструменты для работы. Отладчик GDB

```
(gdb) set disassembly-flavor intel
(gdb) disassemble main
Dump of assembler code for function main:
   0x0000000000401110 <+0>:      mov     eax,0x1
=>  0x0000000000401115 <+5>:      mov     edi,0x1
   0x000000000040111a <+10>:     movabs rsi,0x404028
   0x0000000000401124 <+20>:     mov     edx,0xe
   0x0000000000401129 <+25>:     syscall
   0x000000000040112b <+27>:     mov     eax,0x3c
   0x0000000000401130 <+32>:     mov     edi,0x0
   0x0000000000401135 <+37>:     syscall
   0x0000000000401137 <+39>:     nop     WORD PTR [rax+rax*1+0x0]
End of assembler dump.
```

```
(gdb) break main
Breakpoint 1 at 0x401110: file hello.asm, line 9.
```

```
(gdb) r
Starting program: /home/al/Documents/Education/Дисциплины/АрхитектураBC/2021/presentations/08-IntelAsm/prog/02-hello-nasm/hello

Breakpoint 1, main () at hello.asm:9
9      _      mov     rax, 1      ; 1 = write
```

Инструменты для работы. Отладчик GDB

```
(gdb) list 1,18
1      ; hello.asm
2      section .data
3          msg      db      "Hello, world!",10,0
4      section .bss
5      section .text
6          global main
7      main:
8
9          mov     rax, 1      ; 1 = write
10         mov     rdi, 1      ; 1 = to stdout
11         mov     rsi, msg    ; string to display in rsi
12         mov     rdx, 14     ; length of the string, without 0
13         syscall            ; display the string
14         mov     rax, 60     ; 60 = exit
15         mov     rdi, 0      ; 0 = success exit code
16         syscall            ; quit
17
```

Инструменты для работы. Отладчик GDB

```
(gdb) list 1,18
1      ; hello.asm
2      section .data
3          msg      db      "Hello, world!",10,0
4      section .bss
5      section .text
6          global main
7      main:
8
9          mov     rax, 1      ; 1 = write
10         mov     rdi, 1      ; 1 = to stdout
11         mov     rsi, msg    ; string to display in rs
12         mov     rdx, 14    ; length of the string, w
13         syscall          ; display the string
14         mov     rax, 60    ; 60 = exit
15         mov     rdi, 0    ; 0 = success exit code
16         syscall          ; quit
17
```

```
(gdb) info registers
rax 0x401110 4198672
rbx 0x401140 4198720
rcx 0x7ffff7f74598 140737353565592
rdx 0x7fffffff258 140737488347736
rsi 0x7fffffff248 140737488347720
rdi 0x1 1
rbp 0x0 0x0
rsp 0x7fffffff158 0x7fffffff158
r8 0x0 0
r9 0x7ffff7fdcf0 140737353994192
r10 0x200000000 8589934592
r11 0x206 518
r12 0x401020 4198432
r13 0x0 0
r14 0x0 0
r15 0x0 0
rip 0x401110 0x401110 <main>
eflags 0x246 [ PF ZF IF ]
cs 0x33 51
ss 0x2b 43
ds 0x0 0
es 0x0 0
fs 0x0 0
gs 0x0 0
```

Архитектура x86-64. Регистр флагов

Имя	Символьное обозначение	Бит	Описание
Carry	CF	0	В предыдущей инструкции был выполнен перенос (разрядов)
Parity	PF	2	Последний байт содержит четное число единиц
Adjust	AF	4	Операции BCD (в двоично-десятичном коде)
Zero	ZF	6	Результат предыдущей инструкции равен нулю
Sign	SF	8	В результате выполнения предыдущей инструкции самый значимый бит равен 1
Direction	DF	10	Направление операций со строками (инкремент или декремент)
Overflow	OF	11	В результате выполнения предыдущей инструкции возникло переполнение



Инструменты для работы. Отладчик GDB

```
(gdb) list 1,18
1      ; hello.asm
2      section .data
3          msg      db      "Hello, world!",10,0
4      section .bss
5      section .text
6          global main
7      main:
8
9          mov     rax, 1      ; 1 = write
10         mov     rdi, 1      ; 1 = to stdout
11         mov     rsi, msg    ; string to display in rsi
12         mov     rdx, 14    ; length of the string, with
13         syscall          ; display the string
14         mov     rax, 60    ; 60 = exit
15         mov     rdi, 0      ; 0 = success exit code
16         syscall          ; quit
17
```

```
(gdb) step
10     mov     rdi, 1      ; 1 = to stdout
(gdb) next
11     mov     rsi, msg    ; string to display in rsi
(gdb) n
12     mov     rdx, 14    ; length of the string, with
```

```
(gdb) i r
rax      0x1              1
rbx      0x401140        4198720
rcx      0x7fffffff7f74598 140737353565592
rdx      0x7fffffffefe258 140737488347736
rsi      0x404028        4210728
rdi      0x1              1
rbp      0x0              0x0
rsp      0x7fffffffefe158 0x7fffffffefe158
r8       0x0              0
r9       0x7fffffff7fdcf0 140737353994192
r10      0x200000000      8589934592
r11      0x206            518
r12      0x401020        4198432
r13      0x0              0
r14      0x0              0
r15      0x0              0
rip      0x401124        0x401124 <main+20>
eflags   0x246            [ PF ZF IF ]
cs       0x33            51
ss       0x2b            43
ds       0x0              0
es       0x0              0
fs       0x0              0
gs       0x0              0
```

```
(gdb) delete breakpoints 1
```

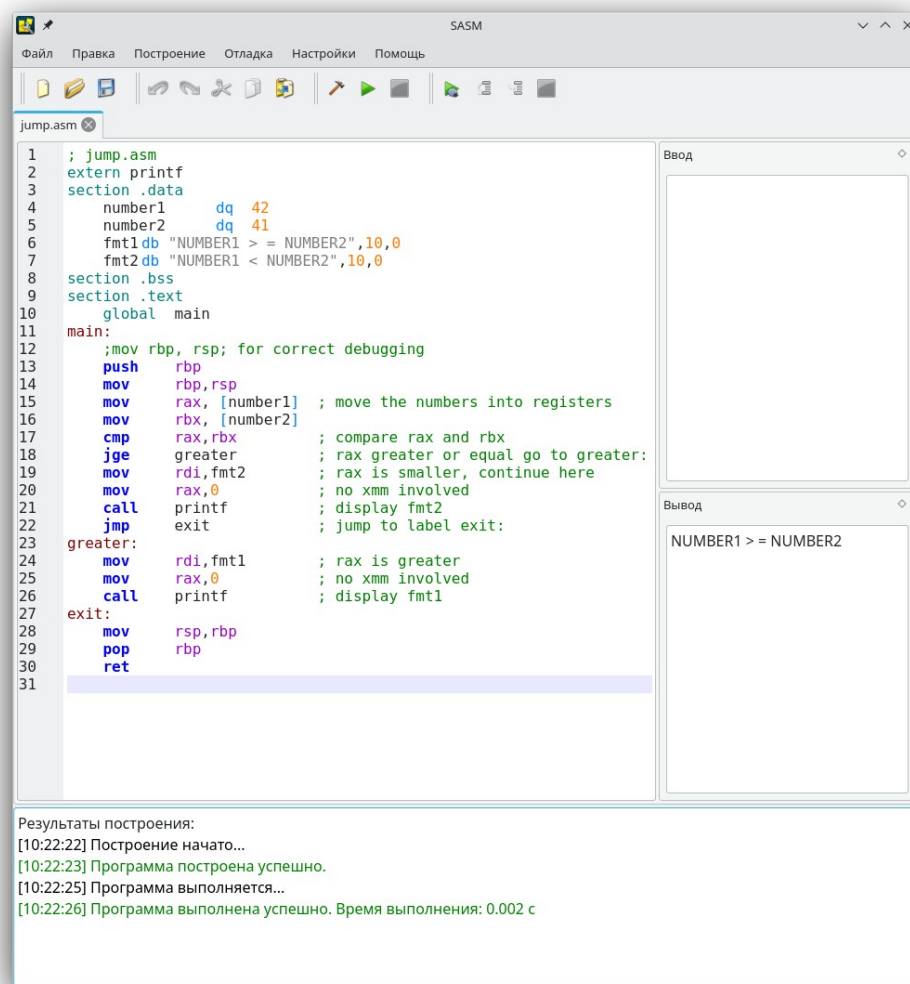
Вычисление длины массива во время ассемблирования

```
1 ; alive.asm
2 section .data
3     msg1    db  "Hello, World!",10,0        ; string with NL and 0
4     msg1Len equ  $-msg1-1                  ; measure the length, minus the 0
5     msg2    db  "Alive and Kicking!",10,0   ; string with NL and 0
6     msg2Len equ  $-msg2-1                  ; measure the length, minus the 0
7     radius  dq  357                          ; no string, not displayable?
8     pi      dq  3.14                          ; no string, not displayable?
9 section .bss
10 section .text
11     global main
12 main:
13     push    rbp                                ; function prologue
14     mov     rbp, rsp                            ; function prologue
15     mov     rax, 1                               ; 1 = write
16     mov     rdi, 1                               ; 1 = to stdout
17     mov     rsi, msg1                           ; string to display
18     mov     rdx, msg1Len                         ; length of the string
19     syscall                                     ; display the string
20     mov     rax, 1                               ; 1 = write
21     mov     rdi, 1                               ; 1 = to stdout
22     mov     rsi, msg2                           ; string to display
23     mov     rdx, msg2Len                         ; length of the string
24     syscall                                     ; display the string
25     mov     rsp, rbp                            ; function epilogue
26     pop     rbp                                ; function epilogue
27     mov     rax, 60                             ; 60 = exit
28     mov     rdi, 0                               ; 0 = success exit code
29     syscall                                     ; quit
```

Использование внешних библиотечных функций

```
; alive.asm
section .data
    msg1      db    "Hello, World!",0
    msg2      db    "Alive and Kicking!",0
    radius    dd    357
    pi        dq    3.14
    fmtstr    db    "%s",10,0 ;format for printing a string
    fmtflt    db    "%lf",10,0 ;format for a float
    fmtint    db    "%d",10,0 ;format for an integer
section .bss
section .text
extern printf
global main
main:
    push     rbp
    mov     rbp, rsp
    mov     rax, 0 ; no floating point
    mov     rdi, fmtstr
    mov     rsi, msg1
    call    printf
    mov     rax, 0 ; no floating point
    mov     rdi, fmtstr
    mov     rsi, msg2
    call    printf
    mov     rax, 0 ; no floating point
    mov     rdi, fmtint
    mov     rsi, [radius]
    call    printf
    mov     rax, 1 ; 1 xmm register used
    movq    xmm0, [pi]
    mov     rdi, fmtflt
    call    printf
    mov     rsp, rbp
    pop     rbp
ret
```

Инструменты для работы. Simple asm (SASM)



The screenshot displays the SASM application window. The main editor contains assembly code for a program named 'jump.asm'. The code defines two data values, 'number1' (42) and 'number2' (41), and compares them. It uses conditional jumps to print the result of the comparison. The output window shows the result: 'NUMBER1 >= NUMBER2'. The status bar at the bottom provides build and execution details.

```
1 ; jump.asm
2 extern printf
3 section .data
4     number1    dq 42
5     number2    dq 41
6     fmt1db "NUMBER1 >= NUMBER2",10,0
7     fmt2db "NUMBER1 < NUMBER2",10,0
8 section .bss
9 section .text
10 global main
11 main:
12     ;mov rbp, rsp; for correct debugging
13     push rbp
14     mov rbp, rsp
15     mov rax, [number1] ; move the numbers into registers
16     mov rbx, [number2]
17     cmp rax, rbx
18     jge greater ; rax greater or equal go to greater:
19     mov rdi, fmt2 ; rax is smaller, continue here
20     mov rax, 0 ; no xmm involved
21     call printf ; display fmt2
22     jmp exit ; jump to label exit:
23 greater:
24     mov rdi, fmt1 ; rax is greater
25     mov rax, 0 ; no xmm involved
26     call printf ; display fmt1
27 exit:
28     mov rsp, rbp
29     pop rbp
30     ret
31
```

Ввод

Выход

NUMBER1 >= NUMBER2

Результаты построения:
[10:22:22] Построение начато...
[10:22:23] Программа построена успешно.
[10:22:25] Программа выполняется...
[10:22:26] Программа выполнена успешно. Время выполнения: 0.002 с

Инструменты для работы. Simple asm (SASM)

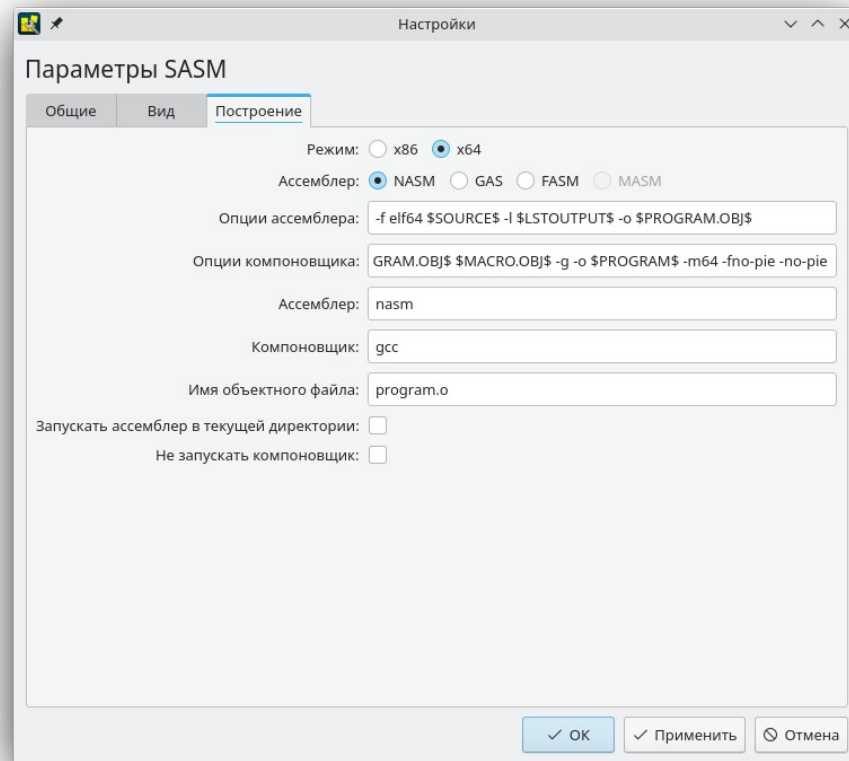
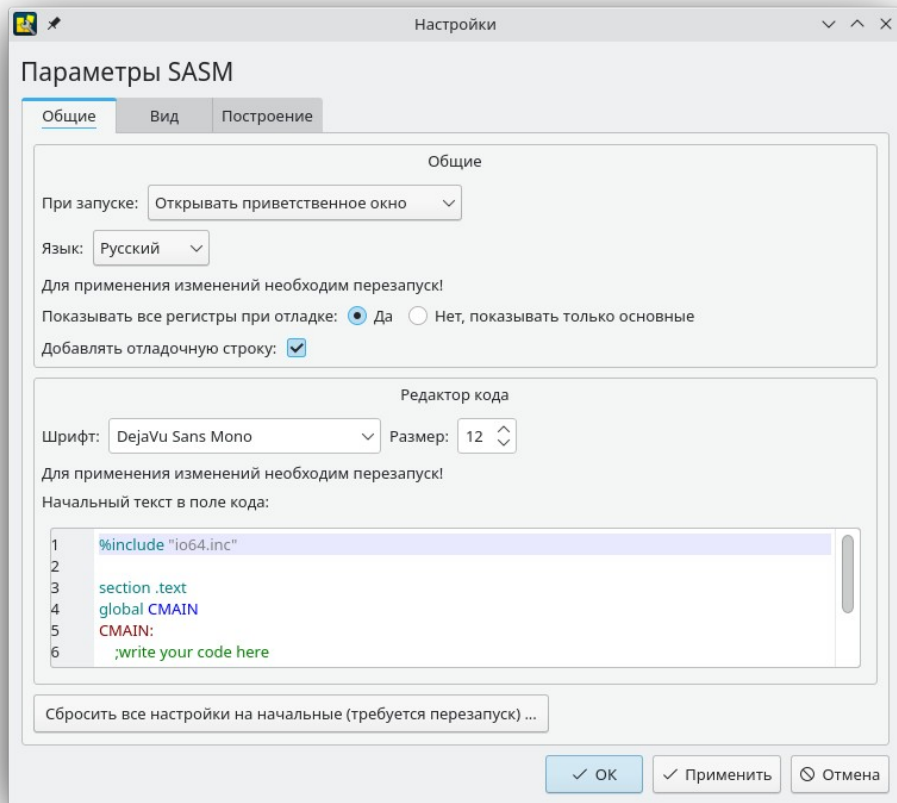
SASM (SimpleASM) - простая кроссплатформенная среда разработки для языков ассемблера NASM, MASM, GAS, FASM с подсветкой синтаксиса и отладчиком. Программа работает "из коробки" и хорошо подойдет для начинающих изучать язык ассемблера. Основана на Qt. Распространяется по свободной лицензии GNU GPL v3.0.

Википедия. SASM: <https://ru.wikipedia.org/wiki/SASM>

SASM – сайт программы: <https://dman95.github.io/SASM/>

Среда программирования SimpleASM. Assembler IDE:
<https://pro-prof.com/forums/topic/simpleasm-assembler-ide>

Инструменты для работы. Simple asm (SASM)



Инструменты для работы. Simple asm (SASM)

The screenshot displays the SASM (Simple Assembler) application window. The main window title is "SASM". The menu bar includes "Файл", "Правка", "Построение", "Отладка", "Настройки", and "Помощь". The toolbar contains icons for file operations and execution. The "Память" (Memory) window is open, showing a table with columns for "Переменная или выражение" (Variable or expression), "Значение" (Value), and "Тип" (Type). The "Память" window also includes a "Добавить..." (Add...) button and a "Smart" dropdown menu. The main editor shows the assembly code for a program named "jump.asm". The code is as follows:

```
1 ; jump.asm
2 extern printf
3 section .data
4     number1     dq 42
5     number2     dq 41
6     fmt1 db "NUMBER1 > = NUMBER2",10,0
7     fmt2 db "NUMBER1 < NUMBER2",10,0
8 section .bss
9 section .text
10 global main
11 main:
12 mov rbp, rsp; for correct debugging
13 ;mov rbp, rsp; for correct debugging
14 push rbp
15 mov rbp, rsp
16 mov rax, [number1] ; move the numbers into registers
17 mov rbx, [number2]
18 cmp rax, rbx ; compare rax and rbx
19 jge greater ; rax greater or equal go to greater:
20 mov rdi, fmt2 ; rax is smaller, continue here
21 mov rax, 0 ; no xmm involved
22 call printf ; display fmt2
23 jmp exit ; jump to label exit:
24 greater:
25 mov rdi, fmt1 ; rax is greater
26 mov rax, 0 ; no xmm involved
27 call printf ; display fmt1
```

The "Ввод" (Input) and "Вывод" (Output) windows are empty. The "Регистры" (Registers) window is also empty, showing a table with columns for "Регистр" (Register), "Hex", and "Info". The status bar at the bottom shows the GDB command line and buttons for "Вывести" (Output) and "Выполнить" (Execute).

[10:27:12] Построение начато...
[10:27:12] Программа построена успешно.
[10:27:12] Отладка началась...
unknown register: Num
unknown register: *

Инструменты для работы. Simple asm (SASM)

The screenshot displays the SASM (Simple Assembler) interface. At the top, there is a menu bar with options: "Файл", "Правка", "Построение", "Отладка", "Настройки", and "Помощь". Below the menu is a toolbar with various icons for file operations and execution.

The main window is divided into several panels:

- Память (Memory):** A table showing memory variables and their types.
- jump.asm:** The assembly code editor with line numbers 1 through 27. Line 18 is highlighted in yellow.
- Ввод (Input):** A panel for entering input data.
- Вывод (Output):** A panel for displaying output.
- Регистры (Registers):** A table showing the current state of CPU registers.
- Log:** A panel at the bottom left showing GDB output messages.
- Команда GDB:** A command line at the bottom for entering GDB commands.

Memory Table:

Переменная или выражение	Значение	Тип	Адрес
number1	{42}	Int q	<input type="checkbox"/>
&fmt1	78'N'	Smart b	<input checked="" type="checkbox"/>
Добавить...		Smart d	<input type="checkbox"/>

Assembly Code (jump.asm):

```
1 ; jump.asm
2 extern printf
3 section .data
4     number1    dq 42
5     number2    dq 41
6     fmt1 db "NUMBER1 > = NUMBER2",10,0
7     fmt2 db "NUMBER1 < NUMBER2",10,0
8 section .bss
9 section .text
10 global main
11 main:
12     mov rbp, rsp; for correct debugging
13     push rbp
14     mov rbp, rsp
15     mov rax, [number1] ; move the numbers into registers
16     mov rbx, [number2]
17     cmp rax, rbx ; compare rax and rbx
18     jge greater ; rax greater or equal go to greater:
19     mov rdi, fmt2 ; rax is smaller, continue here
20     mov rax, 0 ; no xmm involved
21     call printf ; display fmt2
22     jmp exit ; jump to label exit:
23 greater:
24     mov rdi, fmt1 ; rax is greater
25     mov rax, 0 ; no xmm involved
26     call printf ; display fmt1
27 exit:
```

Registers Table:

Регистр	Hex	Value
rax	0x2a	42
rbx	0x29	41
rcx	0x7ffff7f74598	140737353565592
rdx	0x7fffffe598	140737488348568
rsi	0x7fffffe588	140737488348552
rdi	0x1	1
rbp	0x7fffffe490	0x7fffffe490
rsp	0x7fffffe490	0x7fffffe490
r8	0x0	0
r9	0x7ffff7dcfd0	140737353994192
r10	0x7ffff7dc3798	140737351792536
r11	0x206	518
r12	0x401060	4198496
r13	0x0	0
r14	0x0	0
r15	0x0	0
rip	0x40116a	0x40116a <main>
eflags	0x202	[IF]
cs	0x33	51
ss	0x2b	43
ds	0x0	0
es	0x0	0
...

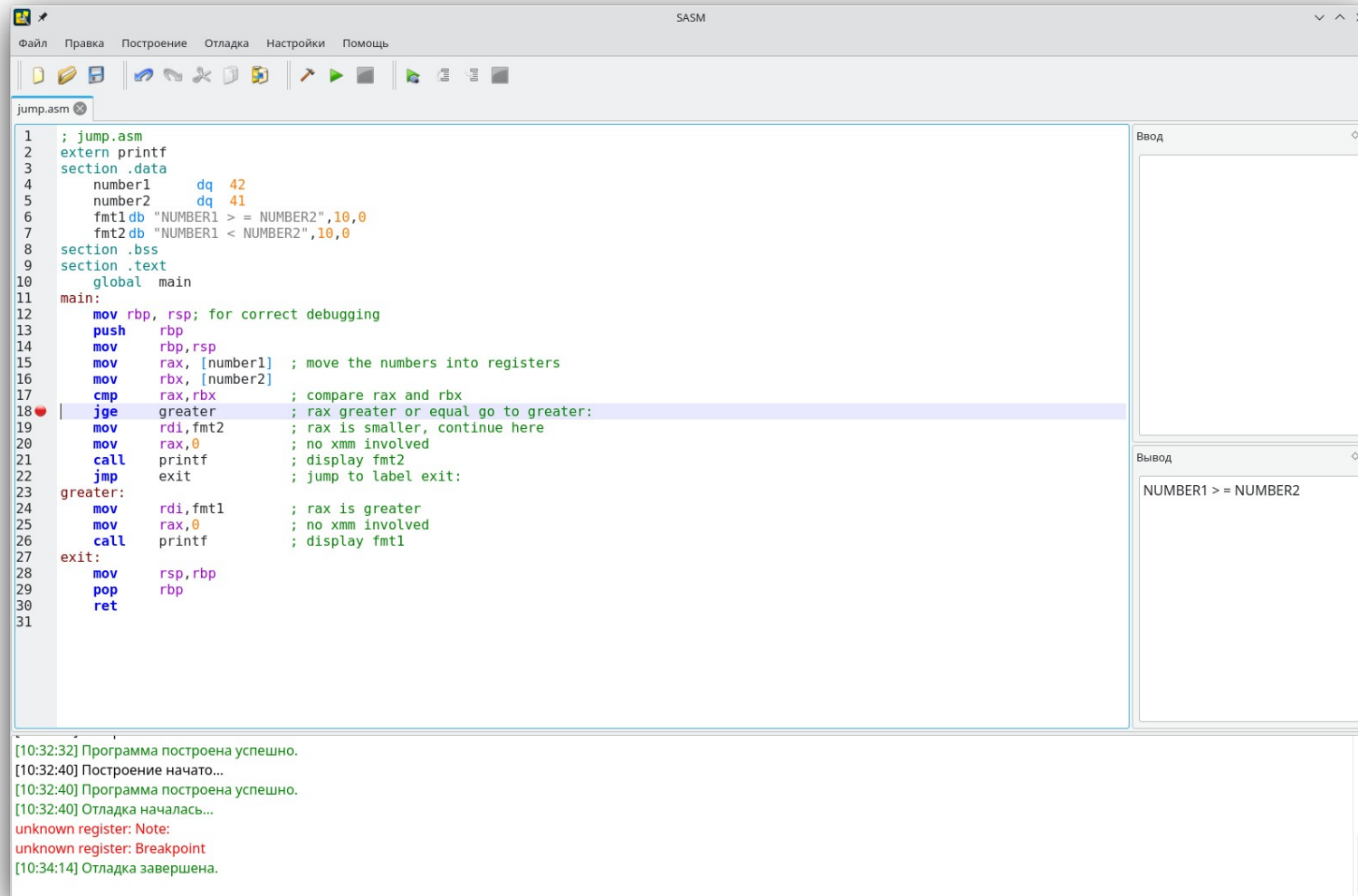
GDB Output:

```
[10:32:40] Построение начато...
[10:32:40] Программа построена успешно.
[10:32:40] Отладка началась...
unknown register: Note:
unknown register: Breakpoint
```

GDB Command Line:

Команда GDB: Вывести

Инструменты для работы. Simple asm (SASM)



The screenshot displays the SASM (Simple Assembler) interface. The main window shows the assembly code for a program named 'jump.asm'. The code defines two numbers, compares them, and prints the result. The output window shows the result: 'NUMBER1 >= NUMBER2'. The status bar at the bottom shows the program was built successfully and the debugger is running.

```
1 ; jump.asm
2 extern printf
3 section .data
4     number1     dq 42
5     number2     dq 41
6     fmt1 db "NUMBER1 >= NUMBER2",10,0
7     fmt2 db "NUMBER1 < NUMBER2",10,0
8 section .bss
9 section .text
10 global main
11 main:
12     mov rbp, rsp; for correct debugging
13     push rbp
14     mov rbp, rsp
15     mov rax, [number1] ; move the numbers into registers
16     mov rbx, [number2]
17     cmp rax, rbx ; compare rax and rbx
18     jge greater ; rax greater or equal go to greater:
19     mov rdi, fmt2 ; rax is smaller, continue here
20     mov rax, 0 ; no xmm involved
21     call printf ; display fmt2
22     jmp exit ; jump to label exit:
23 greater:
24     mov rdi, fmt1 ; rax is greater
25     mov rax, 0 ; no xmm involved
26     call printf ; display fmt1
27 exit:
28     mov rsp, rbp
29     pop rbp
30     ret
31
```

Ввод

Вывод

NUMBER1 >= NUMBER2

[10:32:32] Программа построена успешно.
[10:32:40] Построение начато...
[10:32:40] Программа построена успешно.
[10:32:40] Отладка началась...
unknown register: Note
unknown register: Breakpoint
[10:34:14] Отладка завершена.

Инструменты для работы. Отладчик DDD

```
DDD: /home/al/Projects/beginning-x64-assembly-programming-master/Chapter 07/07_jump/jump.asm
File Edit View Program Commands Status Source Data Help
(): |main
[; jump.asm
extern printf
section .data
    number1 dq      42
    number2 dq      41
    fmt1    db      "NUMBER1 > = NUMBER2",10,0
    fmt2    db      "NUMBER1 < NUMBER2",10,0
section .bss
section .text
global main
main:
    mov     rbp, rsp; for correct debugging
    push   rbp
    mov     rbp, rsp
    mov     rax, [number1] ; move the numbers into registers
    mov     rbx, [number2]
    cmp     rax, rbx      ; compare rax and rbx
    jge     greater      ; rax greater or equal go to greater:
    mov     rdi, fmt2     ; rax is smaller, continue here
    mov     rax, 0        ; no xmm involved
    call   printf        ; display fmt2
    jmp    exit          ; jump to label exit:
greater:
    mov     rdi, fmt1     ; rax is greater
    mov     rax, 0        ; no xmm involved
    call   printf        ; display fmt1
exit:     mov     rsp, rbp
          pop     rbp
          ret

GNU DDD 3.3.12 (x86_64-unknown-linux-gnu), by Dorothea LReading symbols from jump...
(gdb) |
Welcome to DDD 3.3.12 "Dale Head" (x86_64-unknown-linux-gnu)
```

Инструменты для работы. Отладчик DDD

Википередия. Data Display Debugger: https://ru.wikipedia.org/wiki/Data_Display_Debugger

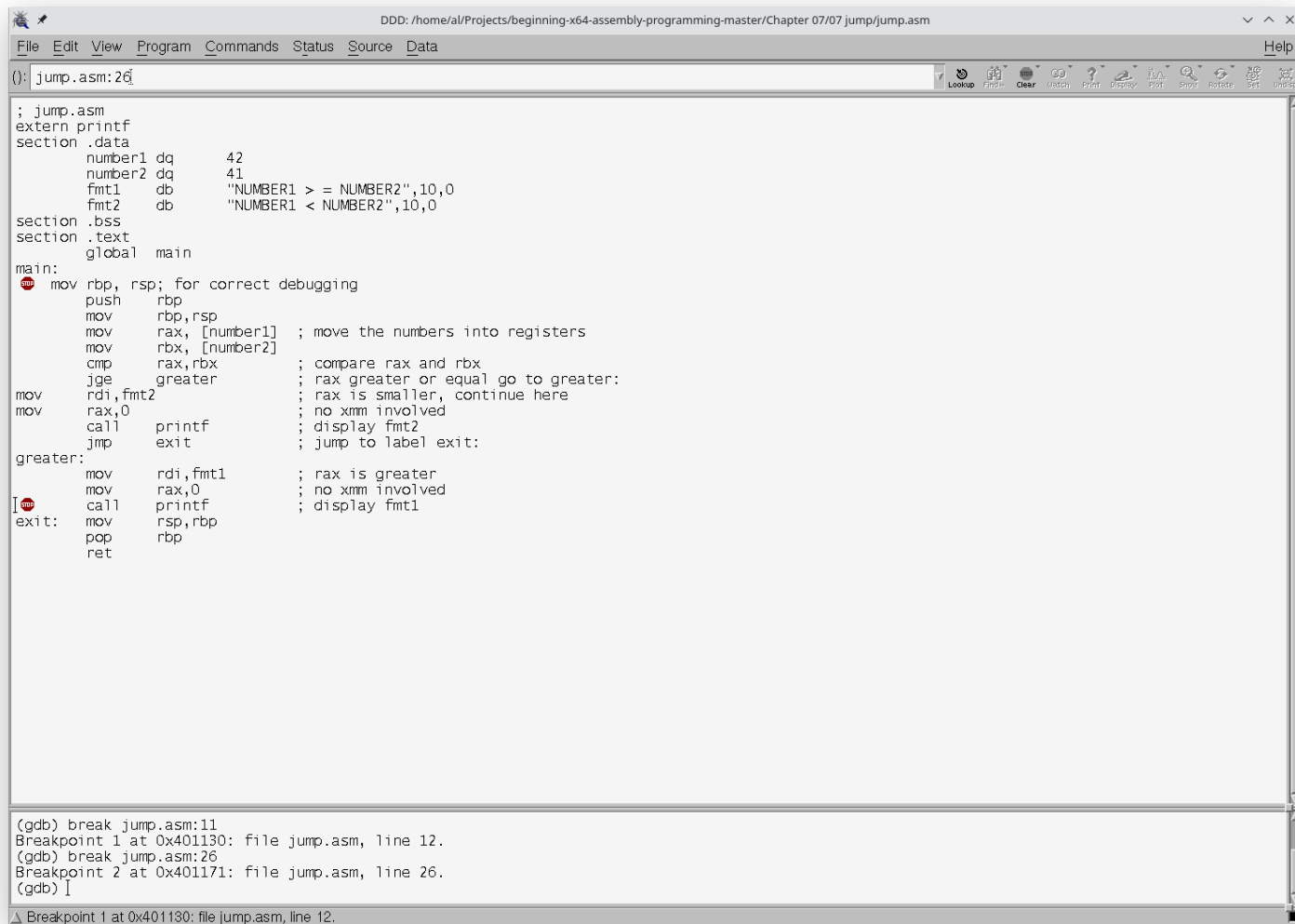
DDD - Data Display Debugger – main page: <https://www.gnu.org/software/ddd/>

DDD - Data Display Debugger - manual: <https://www.gnu.org/software/ddd/manual/>

Самый мощный отладчик во вселенной DDD (Data Display Debugger):
<https://russianblogs.com/article/3755934284/>

Отладка в DDD: http://linux.yaroslavl.ru/docs/altlinux/doc-gnu/ddd/dddm_toc.html

Инструменты для работы. Отладчик DDD



The screenshot shows the DDD (Data Display Debugger) window. The title bar indicates the file path: DDD: /home/al/Projects/beginning-x64-assembly-programming-master/Chapter 07/07 jump/jump.asm. The menu bar includes File, Edit, View, Program, Commands, Status, Source, Data, and Help. The main window displays the assembly code for 'jump.asm' with line numbers 1 through 26. The code includes sections for .data, .bss, and .text, and a main function with various assembly instructions and comments. The GDB console at the bottom shows the following commands and output:

```
(gdb) break jump.asm:11
Breakpoint 1 at 0x401130: file jump.asm, line 12.
(gdb) break jump.asm:26
Breakpoint 2 at 0x401171: file jump.asm, line 26.
(gdb) |
^
Breakpoint 1 at 0x401130: file jump.asm, line 12.
```


Инструменты для работы. Отладчик DDD

The screenshot displays the DDD (Debugging Dynamic Data) debugger interface. The window title is "DDD: /home/al/Projects/beginning-x64-assembly-programming-master/Chapter 07/07_jump/jump.asm". The menu bar includes File, Edit, View, Program, Commands, Status, Source, Data, and Help. The main window shows the assembly source code for "jump.asm".

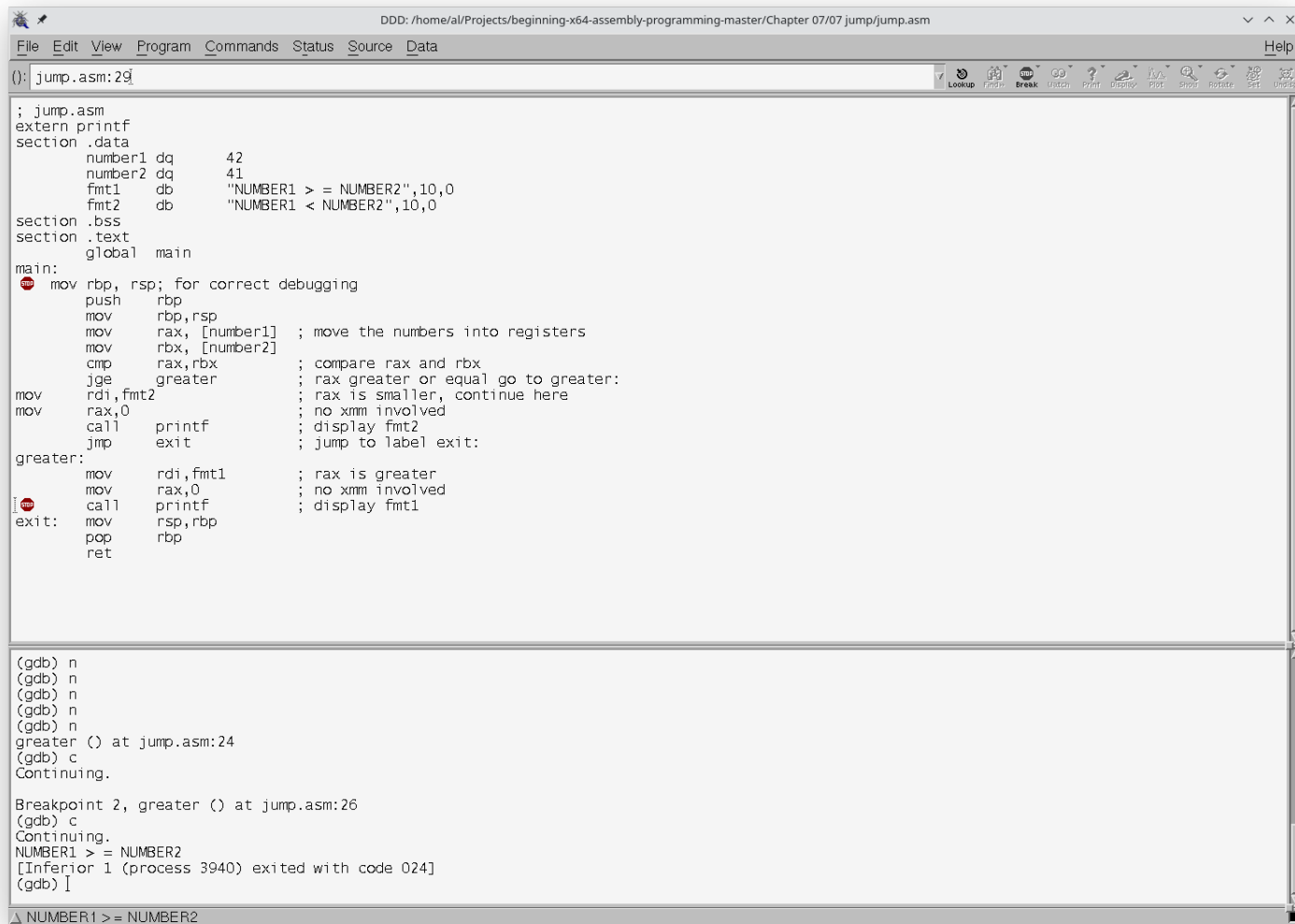
```
; jump.asm
extern printf
section .data
    number1 dq      42
    number2 dq      41
    fmt1    db      "NUMBER1 > = NUMBER2",10,0
    fmt2    db      "NUMBER1 < NUMBER2",10,0
section .bss
section .text
    global  main
main:
    mov    rbp, rsp; for correct debugging
    push  rbp
    mov   rbp, rsp
    mov   rax, [number1] ; move the numbers into registers
    mov   rbx, [number2]
    cmp   rax, rbx      ; compare rax and rbx
    jge   greater      ; rax greater or equal go to greater:
    mov   rdi, fmt2     ; rax is smaller, continue here
    mov   rax, 0        ; no xmm involved
    call  printf        ; display fmt2
    jmp   exit          ; jump to label exit:
greater:
    mov   rdi, fmt1     ; rax is greater
    mov   rax, 0        ; no xmm involved
    call  printf        ; display fmt1
exit:   mov   rsp, rbp
        pop   rbp
        ret
```

The bottom pane shows the disassembly of the main function:

```
(gdb) disassemble main
Dump of assembler code for function main:
0x0000000000401130 <+0>:  mov    rbp, rsp
0x0000000000401133 <+3>:  push  rbp
0x0000000000401134 <+4>:  mov   rbp, rsp
0x0000000000401137 <+7>:  mov   rax, QWORD PTR ds:0x404030
0x000000000040113f <+15>: mov   rbx, QWORD PTR ds:0x404038
0x0000000000401147 <+23>: cmp   rax, rbx
0x000000000040114a <+26>: jge   0x401162 <greater>
0x000000000040114c <+28>: movabs rdi, 0x404055
0x0000000000401156 <+38>: mov   eax, 0
0x000000000040115b <+43>: call  0x401030 <printf@plt>
0x0000000000401160 <+48>: jmp   0x401176 <exit>
End of assembler dump.
(gdb)
```

At the bottom of the window, there is a status bar with the text: "▲ Dump of assembler code for function main:".

Инструменты для работы. Отладчик DDD



The screenshot shows the DDD debugger window. The title bar reads "DDD: /home/al/Projects/beginning-x64-assembly-programming-master/Chapter 07/07 jump/jump.asm". The menu bar includes File, Edit, View, Program, Commands, Status, Source, Data, and Help. The toolbar contains icons for various debugging actions like Lookup, Find, Break, Watch, Print, Disasm, Step, Step In, Step Out, Rotate, and Undo. The main window displays the assembly code for "jump.asm" with several lines highlighted in red, indicating breakpoints. The GDB console at the bottom shows the execution flow, including a breakpoint hit at line 26 and the program's exit.

```
; jump.asm
extern printf
section .data
    number1 dq    42
    number2 dq    41
    fmt1    db    "NUMBER1 > = NUMBER2",10,0
    fmt2    db    "NUMBER1 < NUMBER2",10,0
section .bss
section .text
global main
main:
mov rbp, rsp; for correct debugging
push rbp
mov rbp, rsp
mov rax, [number1] ; move the numbers into registers
mov rbx, [number2]
cmp rax, rbx      ; compare rax and rbx
jge greater      ; rax greater or equal go to greater:
mov rdi, fmt2     ; rax is smaller, continue here
mov rax, 0        ; no xmm involved
call printf       ; display fmt2
jmp exit         ; jump to label exit:
greater:
mov rdi, fmt1     ; rax is greater
mov rax, 0        ; no xmm involved
call printf       ; display fmt1
exit:
mov rsp, rbp
pop rbp
ret

(gdb) n
(gdb) n
(gdb) n
(gdb) n
(gdb) n
(gdb) n
greater () at jump.asm:24
(gdb) c
Continuing.

Breakpoint 2, greater () at jump.asm:26
(gdb) c
Continuing.
NUMBER1 > = NUMBER2
[Inferior 1 (process 3940) exited with code 024]
(gdb) ]
```

△ NUMBER1 > = NUMBER2