Лабораторная работа №9. Понятие подпрограммы. Отладчик GDB.

Титульный лист

Дисциплина: Архитектура ЭВМ
Лабораторная работа №9: Понятие подпрограммы. Отладчик GDB.
ФИО студента: Сако Лассине
Группа: НПИБД-02-25
Дата выполнения: 2025 год

---

1. Цель работы

Приобретение навыков написания программ с использованием подпрограмм. Знакомство с методами отладки при помощи GDB и его основными возможностями.

2. Результаты выполнения лабораторной работы

2.1. Программа lab09-1.asm

Задание: Реализация программы вычисления арифметического выражения с использованием подпрограммы.

Код программы: “‘asm %include ‘in_out.asm’

SECTION .data msg: DB ‘Введите x:’,0 result: DB ‘2x+7=’,0

SECTION .bss x: RESB 80 res: RESB 80

SECTION .text GLOBAL _start _start:

;— ; Основная программа ;— mov eax, msg call sprint

```
mov ecx, x
mov edx, 80
call sread

mov eax,x
call atoi

call _calcul    ;            _calcul

mov eax,result
```

```
call sprint
mov eax,[res]
call iprintLF

call quit
```

;— ; Подпрограмма вычисления ; выражения “2x+7” _calcul: mov ebx,2 mul ebx add eax,7 mov [res],eax ret ; выход из подпрограммы

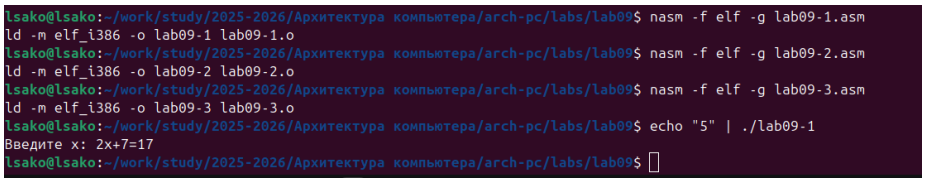

Figure 1: Compilation et test des programmes

2.2. Программа lab09-2.asm

Задание: Создание программы “Hello World” для изучения отладчика GDB.

Код программы: “‘asm SECTION .data msg1: db “Hello,”,0 msg1Len: equ $ - msg1

```
msg2:    db "world!",0xa
msg2Len:   equ $ - msg2
```

SECTION .text global _start

_start: mov eax, 4 mov ebx, 1 mov ecx, msg1 mov edx, msg1Len int 0x80

```
mov eax, 4
mov ebx, 1
mov ecx, msg2
mov edx, msg2Len
int 0x80

mov eax, 1
mov ebx, 0
int 0x80
```

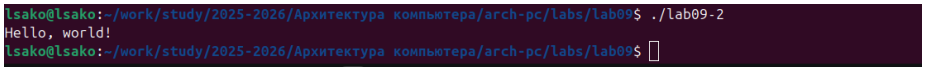

Figure 2: Test Hello World

2.3. Работа с отладчиком GDB

Задание: Изучение основных возможностей отладчика GDB.

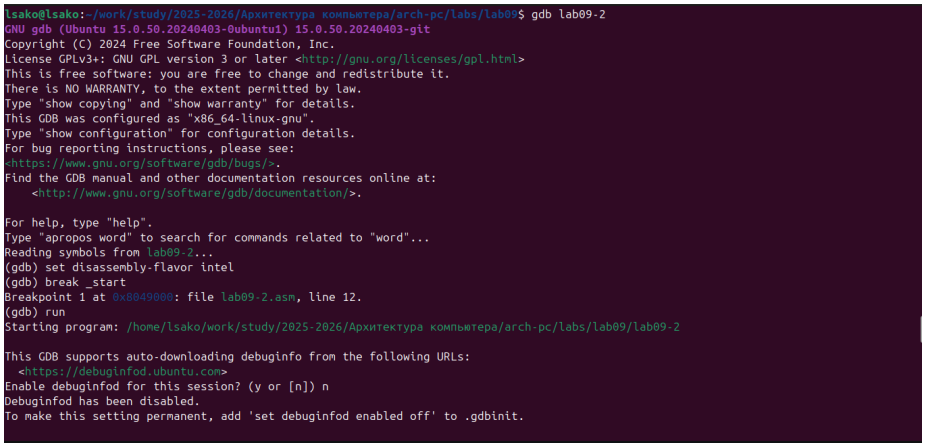

Figure 3: Début de session GDB lab09-2

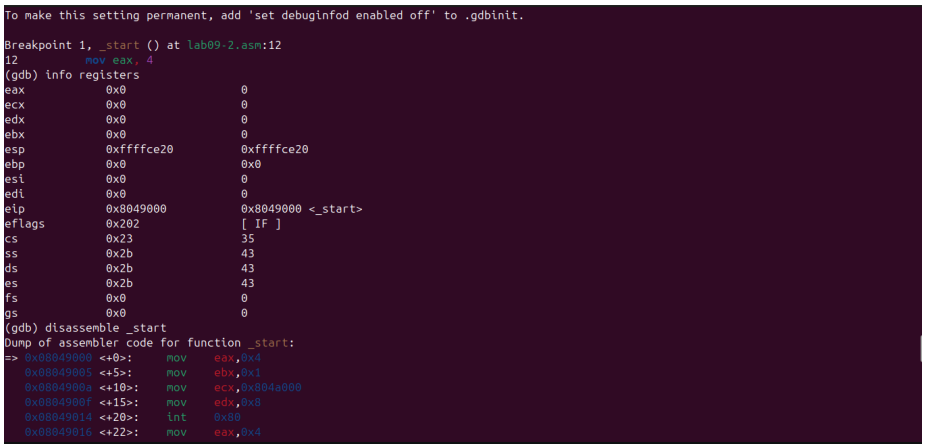

Figure 4: Registres initiaux dans GDB

2.4. Программа lab09-3.asm

Задание: Создание программы для обработки аргументов командной строки.

Код программы: “‘asm SECTION .data newline: db 0xA

SECTION .text global _start

```
   0x08049016 <+22>:    mov    eax,0x4
   0x0804901b <+27>:    mov    ebx,0x1
   0x08049020 <+32>:    mov    ecx,0x804a008
   0x08049025 <+37>:    mov    edx,0x7
   0x0804902a <+42>:    int    0x80
   0x0804902c <+44>:    mov    eax,0x1
   0x08049031 <+49>:    mov    ebx,0x0
   0x08049036 <+54>:    int    0x80
End of assembler dump.
(gdb) stepi
13          mov ebx, 1
(gdb) info registers
eax            0x4                 4
ecx            0x0                 0
edx            0x0                 0
ebx            0x0                 0
esp            0xffffce20          0xffffce20
ebp            0x0                 0x0
esi            0x0                 0
edi            0x0                 0
eip            0x8049005           0x8049005 <_start+5>
eflags         0x202               [ IF ]
cs             0x23                35
ss             0x2b                43
ds             0x2b                43
es             0x2b                43
fs             0x0                 0
gs             0x0                 0
```

Figure 5: Exécution pas-à-pas GDB

```
fs             0x0                 0
gs             0x0                 0
(gdb) x/1sb 0x804a000
0x804a000 <msg1>:       "Hello, "
(gdb) stepi
14          mov ecx, msg1
(gdb) stepi
15          mov edx, msg1Len
(gdb) stepi
16          int 0x80
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:       "world!\n\034"
(gdb) info breakpoints
Num     Type           Disp Enb Address    What
1       breakpoint     keep y   0x08049000 lab09-2.asm:12
        breakpoint already hit 1 time
(gdb) set {char}0x804a000='h'
(gdb) x/1sb 0x804a000
0x804a000 <msg1>:       "hello, "
(gdb) set {char}0x804a008='W'
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:       "World!\n\034"
(gdb) print/x $eax
$1 = 0x4
(gdb) print/t $eax
$2 = 100
(gdb) print/d $eax
$3 = 4
(gdb) set $ebx=2
```

Figure 6: Modification de la mémoire en GDB

```
(gdb) print/d $eax
$3 = 4
(gdb) set $ebx=2
(gdb) print/d $ebx
$4 = 2
(gdb) continue
Continuing.
hello, World!
[Inferior 1 (process 6351) exited normally]
(gdb) (gdb) quit
```

Figure 7: Résultat des modifications GDB

_start: ; Получаем количество аргументов pop eax ; argc

```
;
pop ebx          ; argv[0]
```

print_args: ; Печатаем аргумент pop ebx ; следующий аргумент test ebx, ebx jz exit

```
;
mov ecx, ebx
call strlen

;
mov edx, eax    ;
mov eax, 4      ; sys_write
mov ebx, 1      ; stdout
int 0x80

;
mov eax, 4
mov ebx, 1
mov ecx, newline
mov edx, 1
int 0x80

jmp print_args
```

exit: mov eax, 1 ; sys_exit mov ebx, 0 ; код выхода int 0x80

; Функция вычисления длины строки ; Вход: ecx = адрес строки ; Выход: eax = длина strlen: push ecx mov eax, 0 .count_loop: cmp byte [ecx], 0 je .done inc ecx inc eax jmp .count_loop .done: pop ecx ret

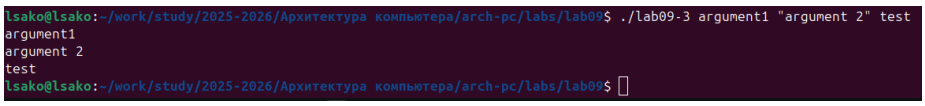

Figure 8: Test des arguments sans GDB

2.5. Анализ аргументов командной строки в GDB

Задание: Исследование расположения аргументов командной строки в стеке.

```
lsako@lsako:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab09$ gdb --args lab09-3 argument1 "argument 2" test
GNU gdb (Ubuntu 15.0.50.20240403-0ubuntu1) 15.0.50.20240403-git
Copyright (C) 2024 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab09-3...
(gdb) set disassembly-flavor intel
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-3.asm, line 9.
(gdb) run
Starting program: /home/lsako/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab09/lab09-3 argument1 argument\ 2 test

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.ubuntu.com>
Enable debuginfod for this session? (y or [n]) n
Debuginfod has been disabled.
```

Figure 9: Lancement GDB avec arguments

```
Debuginfod has been disabled.
To make this setting permanent, add 'set debuginfod enabled off' to .gdbinit.

Breakpoint 1, _start () at lab09-3.asm:9
9           pop eax         ; argc
(gdb) x/x $esp
0xffffcdf0:     0x00000004
(gdb) x/s *(void**)($esp + 4)
0xffffcfe8:     "/home/lsako/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab09/lab09-3"
(gdb) x/s *(void**)($esp + 8)
0xffffd050:     "argument1"
(gdb) x/s *(void**)($esp + 12)
0xffffd05a:     "argument 2"
(gdb) x/s *(void**)($esp + 16)
0xffffd068:     "test"
(gdb) print/d *(int*)($esp)
$1 = 4
(gdb) continue
Continuing.
argument1
argument 2
test
[Inferior 1 (process 6785) exited normally]
(gdb) quit
lsako@lsako:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab09$
```

Figure 10: Analyse de la pile des arguments

```
lsako@lsako:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab09$ file lab09-2
lab09-2: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), statically linked, with debug_info, not stripped
lsako@lsako:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab09$
```

Figure 11: Vérification des informations de débogage

2.6. Проверка отладочной информации

3. Результаты выполнения заданий для самостоятельной работы

3.1. Модификация программы lab09-1.asm

Задание: Добавление подпрограммы _subcalcul в подпрограмму _calcul для вычисления выражения f(g(x)).

Модифицированный код: “‘asm %include ‘in_out.asm’

SECTION .data msg: DB ‘Введите x:’,0 result: DB ’f(g(x))=2*(3x-1)+7=’,0

SECTION .bss x: RESB 80 res: RESB 80 temp: RESB 80

SECTION .text GLOBAL _start _start: mov eax, msg call sprint

```
mov ecx, x
mov edx, 80
call sread

mov eax,x
call atoi

call _calcul

mov eax,result
call sprint
mov eax,[res]
call iprintLF

call quit
```

;— ; Подпрограмма вычисления f(g(x)) _calcul: push eax ; сохраняем x call _subcalcul ; вычисляем g(x)=3x-1 mov ebx,2 mul ebx ; 2*g(x) add eax,7 ; 2*g(x)+7 mov [res],eax pop eax ; восстанавливаем стек ret

;— ; Подпрограмма вычисления g(x)=3x-1 _subcalcul: mov ebx,3 mul ebx ; 3*x sub eax,1 ; 3*x-1 ret

3.2. Отладка программы с ошибкой

Задание: Найти и исправить ошибку в программе вычисления выражения (3+2)*4+5 с помощью GDB.

Исправленный код: “‘asm %include ‘in_out.asm’

SECTION .data div: DB ‘Результат:’,0

SECTION .text GLOBAL _start _start: ; — Вычисление выражения (3+2)*4+5 mov eax,3 mov ebx,2 add eax,ebx ; (3+2)=5 mov ebx,4 mul ebx ; 5*4=20 add eax,5 ; 20+5=25 mov edi,eax

; — Вывод результата на экран mov eax,div call sprint mov eax,edi call iprintLF call quit

4. Ответы на вопросы для самопроверки

1. **Какие языковые средства используются в ассемблере для оформления и активизации подпрограмм?**
2. **Объясните механизм вызова подпрограмм.**
3. **Как используется стек для обеспечения взаимодействия между вызывающей и вызываемой процедурами?**
4. **Каково назначение операнда в команде ret?**
5. **Для чего нужен отладчик?**
6. **Объясните назначение отладочной информации и как нужно компилировать программу, чтобы в ней присутствовала отладочная информация.**
7. **Расшифруйте и объясните следующие термины: breakpoint, watchpoint, checkpoint, catchpoint и call stack.**
8. **Назовите основные команды отладчика gdb и как они могут быть использованы для отладки программ.**

5. Выводы

В ходе выполнения лабораторной работы:

1. **Освоено программирование с использованием подпрограмм** - изучены инструкции call и ret, созданы программы с многоуровневыми вызовами подпрограмм.
2. **Приобретены навыки работы с отладчиком GDB** - освоены команды установки точек останова, пошагового выполнения, анализа регистров и памяти.
3. **Изучены методы отладки программ** - практически применены различные подходы к поиску и исправлению ошибок.
4. **Освоена работа с аргументами командной строки** - исследовано расположение параметров в стеке памяти.

5. **Практически применены навыки модификации программ** - успешно добавлены дополнительные подпрограммы и исправлены ошибки в существующем коде.
6. **Изучены возможности отладочной информации** - освоена компиляция программ с ключом -g для эффективной отладки.

Цель работы достигнута - приобретены практические навыки написания программ с подпрограммами и использования отладчика GDB для анализа и отладки кода.