Министерство цифрового развития, связи и
массовых коммуникаций Российской Федерации
СибГУТИ

Отчёт по информатике
«Генерация и анализ ассемблерного кода программы на Си под разные архитектуры»

Выполнил студент: Родионов П.А.
группы: ИВ-522
Вариант 9
Проверил ассистент кафедры ВС: Шевченко М.В.

Новосибирск 2025 г.

1. Исходный код

```
void inc(int *p){ (*p)++; }
void dec(int *p){ (*p)--; }

int main(){
    int x = 10;
    inc(&x);
    dec(&x);
    return 0;
}
```

---

2. Процесс получения ассемблерных листингов

2.1. ОС и системная информация

Linux

```
Linux archlinux 6.17.8-arch1-1 x86_64 GNU/Linux
```

MinGW (Windows x86-64, через winboat)

Компилятор: x86_64-w64-mingw32-gcc

RISC-V (кросс-компиляция)

Использован пакет: `riscv64-linux-gnu-gcc`

2.2. Версии компиляторов

GCC Linux

```
gcc (GCC) 15.2.1 20251112
```

MinGW x86-64

```
x86_64-w64-mingw32-gcc (GCC) 15.2.0
```

RISC-V GCC

```
riscv64-linux-gnu-gcc (GCC) 15.1.0
```

2.3. Команды компиляции

Linux x86-64

```
gcc -S -O0 main.c -o linux_x86.s
```

Windows x86-64 (MinGW-w64)

```
x86_64-w64-mingw32-gcc -S -O0 main.c -o win_x86.s
```

RISC-V 64

```
riscv64-linux-gnu-gcc -S -O0 main.c -o riscv.s
```

2.4. Установка RISC-V компилятора

На Arch Linux:

```
sudo pacman -S riscv64-linux-gnu-gcc
```

Это добавляет полный кросс-компилятор и позволяет компилировать C-программы в RISC-V ELF.

---

3. Ассемблерные листинги

3.1. Linux x86-64 (GCC 15.2.1, -O0)

```
1	.file	"main.c"
2	.text
```

```
3           .globl  inc
4           .type   inc, @function
5   inc:
6   .LFB0:
7           .cfi_startproc
8           pushq  %rbp
9           .cfi_def_cfa_offset 16
10          .cfi_offset 6, -16
11          movq   %rsp, %rbp
12          .cfi_def_cfa_register 6
13          movq   %rdi, -8(%rbp)
14          movq   -8(%rbp), %rax
15          movl    (%rax), %eax
16          leal    1(%rax), %edx
17          movq   -8(%rbp), %rax
18          movl    %edx, (%rax)
19          nop
20          popq    %rbp
21          .cfi_def_cfa 7, 8
22          ret
23          .cfi_endproc
24  .LFE0:
25          .size   inc, .-inc
26          .globl  dec
27          .type   dec, @function
28  dec:
29  .LFB1:
30          .cfi_startproc
31          pushq  %rbp
32          .cfi_def_cfa_offset 16
33          .cfi_offset 6, -16
34          movq   %rsp, %rbp
35          .cfi_def_cfa_register 6
36          movq   %rdi, -8(%rbp)
37          movq   -8(%rbp), %rax
38          movl    (%rax), %eax
39          leal    -1(%rax), %edx
40          movq   -8(%rbp), %rax
41          movl    %edx, (%rax)
42          nop
43          popq    %rbp
44          .cfi_def_cfa 7, 8
45          ret
46          .cfi_endproc
```

```
47  .LFE1:
48          .size   dec, .-dec
49          .globl  main
50          .type   main, @function
51  main:
52  .LFB2:
53          .cfi_startproc
54          pushq  %rbp
55          .cfi_def_cfa_offset 16
56          .cfi_offset 6, -16
57          movq   %rsp, %rbp
58          .cfi_def_cfa_register 6
59          subq   $16, %rsp
60          movq   %fs:40, %rax
61          movq   %rax, -8(%rbp)
62          xorl   %eax, %eax
63          movl   $10, -12(%rbp)
64          leaq   -12(%rbp), %rax
65          movq   %rax, %rdi
66          call   inc
67          leaq   -12(%rbp), %rax
68          movq   %rax, %rdi
69          call   dec
70          movl   $0, %eax
71          movq   -8(%rbp), %rdx
72          subq   %fs:40, %rdx
73          je     .L5
74          call   __stack_chk_fail@PLT
75  .L5:
76          leave
77          .cfi_def_cfa 7, 8
78          ret
79          .cfi_endproc
80  .LFE2:
81          .size   main, .-main
82          .ident  "GCC: (GNU) 15.2.1 20251112"
83          .section        .note.GNU-stack,"",@progbits
```

3.2. Windows x86-64 (MinGW GCC 15.2.0, -O0)

```
1           .file   "main.c"
2           .text
3           .globl  inc
```

```
4           .def    inc;    .scl    2;      .type   32;     .endef
5           .seh_proc       inc
6   inc:
7           pushq   %rbp
8           .seh_pushreg    %rbp
9           movq    %rsp, %rbp
10          .seh_setframe %rbp, 0
11          .seh_endprologue
12          movq    %rcx, 16(%rbp)
13          movq    16(%rbp), %rax
14          movl    (%rax), %eax
15          leal    1(%rax), %edx
16          movq    16(%rbp), %rax
17          movl    %edx, (%rax)
18          nop
19          popq    %rbp
20          ret
21          .seh_endproc
22          .globl  dec
23          .def    dec;    .scl    2;      .type   32;     .endef
24          .seh_proc       dec
25  dec:
26          pushq   %rbp
27          .seh_pushreg    %rbp
28          movq    %rsp, %rbp
29          .seh_setframe %rbp, 0
30          .seh_endprologue
31          movq    %rcx, 16(%rbp)
32          movq    16(%rbp), %rax
33          movl    (%rax), %eax
34          leal    -1(%rax), %edx
35          movq    16(%rbp), %rax
36          movl    %edx, (%rax)
37          nop
38          popq    %rbp
39          ret
40          .seh_endproc
41          .globl  main
42          .def    main;   .scl    2;      .type   32;     .endef
43          .seh_proc       main
44  main:
45          pushq   %rbp
46          .seh_pushreg    %rbp
47          movq    %rsp, %rbp
```

```
48              .seh_setframe %rbp, 0
49              subq    $48, %rsp
50              .seh_stackalloc         48
51              .seh_endprologue
52              call    __main
53              movl    $10, -4(%rbp)
54              leaq    -4(%rbp), %rax
55              movq    %rax, %rcx
56              call    inc
57              leaq    -4(%rbp), %rax
58              movq    %rax, %rcx
59              call    dec
60              movl    $0, %eax
61              addq    $48, %rsp
62              popq    %rbp
63              ret
64              .seh_endproc
65              .def    __main;         .scl    2;      .type   32;     .endef
66              .ident  "GCC: (GNU) 15.2.0"
```

4.3. RISC-V (riscv64-linux-gnu-gcc, -O0)

```
1               .file   "main.c"
2               .option pic
3               .attribute arch,
"rv64i2p1_m2p0_a2p1_f2p2_d2p2_c2p0_zicsr2p0_zifencei2p0_zmmul1p0_zaamo1p0_zalrsc1
p0_zca1p0_zcd1p0"
4               .attribute unaligned_access, 0
5               .attribute stack_align, 16
6               .text
7               .align  1
8               .globl  inc
9               .type   inc, @function
10      inc:
11      .LFB0:
12              .cfi_startproc
13              addi    sp,sp,-32
14              .cfi_def_cfa_offset 32
15              sd      ra,24(sp)
16              sd      s0,16(sp)
17              .cfi_offset 1, -8
18              .cfi_offset 8, -16
19              addi    s0,sp,32
```

```
20              .cfi_def_cfa 8, 0
21              sd      a0,-24(s0)
22              ld      a5,-24(s0)
23              lw      a5,0(a5)
24              addiw   a5,a5,1
25              sext.w  a4,a5
26              ld      a5,-24(s0)
27              sw      a4,0(a5)
28              nop
29              ld      ra,24(sp)
30              .cfi_restore 1
31              ld      s0,16(sp)
32              .cfi_restore 8
33              .cfi_def_cfa 2, 32
34              addi    sp,sp,32
35              .cfi_def_cfa_offset 0
36              jr      ra
37              .cfi_endproc
38      .LFE0:
39              .size   inc, .-inc
40              .align  1
41              .globl  dec
42              .type   dec, @function
43      dec:
44      .LFB1:
45              .cfi_startproc
46              addi    sp,sp,-32
47              .cfi_def_cfa_offset 32
48              sd      ra,24(sp)
49              sd      s0,16(sp)
50              .cfi_offset 1, -8
51              .cfi_offset 8, -16
52              addi    s0,sp,32
53              .cfi_def_cfa 8, 0
54              sd      a0,-24(s0)
55              ld      a5,-24(s0)
56              lw      a5,0(a5)
57              addiw   a5,a5,-1
58              sext.w  a4,a5
59              ld      a5,-24(s0)
60              sw      a4,0(a5)
61              nop
62              ld      ra,24(sp)
63              .cfi_restore 1
```

```
64              ld      s0,16(sp)
65              .cfi_restore 8
66              .cfi_def_cfa 2, 32
67              addi    sp,sp,32
68              .cfi_def_cfa_offset 0
69              jr      ra
70              .cfi_endproc
71      .LFE1:
72              .size   dec, .-dec
73              .align  1
74              .globl  main
75              .type   main, @function
76      main:
77      .LFB2:
78              .cfi_startproc
79              addi    sp,sp,-32
80              .cfi_def_cfa_offset 32
81              sd      ra,24(sp)
82              sd      s0,16(sp)
83              .cfi_offset 1, -8
84              .cfi_offset 8, -16
85              addi    s0,sp,32
86              .cfi_def_cfa 8, 0
87              li      a5,10
88              sw      a5,-20(s0)
89              addi    a5,s0,-20
90              mv      a0,a5
91              call    inc
92              addi    a5,s0,-20
93              mv      a0,a5
94              call    dec
95              li      a5,0
96              mv      a0,a5
97              ld      ra,24(sp)
98              .cfi_restore 1
99              ld      s0,16(sp)
100             .cfi_restore 8
101             .cfi_def_cfa 2, 32
102             addi    sp,sp,32
103             .cfi_def_cfa_offset 0
104             jr      ra
105             .cfi_endproc
106     .LFE2:
107             .size   main, .-main
```

```
108             .ident  "GCC: (GNU) 15.1.0"
109             .section        .note.GNU-stack,"",@progbits
```

---

5. Анализ и сравнение архитектур

5.1. Различия в регистрах

Архитектура	Регистры	Комментарий
x86-64 Linux	`%rax %rbp %rsp %rdi ...`	System V ABI: первый аргумент в %rdi
x86-64 Windows	`%rcx %rdx %r8 %r9`	Microsoft x64 ABI: первый аргумент в %rcx
RISC-V 64	`a0–a7`, `s0`, `ra`	Полностью load/store архитектура

Ключевое отличие:

- Linux кладёт аргумент `inc(int *p)` в **%rdi**
- Windows кладёт аргумент в **%rcx**
- RISC-V кладёт аргумент в **a0**

5.2. Стек, пролог и эпилог

Linux x86-64

```
pushq %rbp
mov %rsp, %rbp
...
popq %rbp
```

Windows x86-64

Добавляются SEH-директивы (`.seh_pushreg`, `.seh_endprologue`) — это механизм Structured Exception Handling.

RISC-V

```
addi sp, sp, -32
sd ra, 24(sp)
sd s0, 16(sp)
...
addi sp, sp, 32
```

RISC-V всегда явно сохраняет регистры, т.к. регистров больше и ABI строже.

5.3. Инструкции инкремента / декремента

x86-64

Используется эффективная LEA:

```
leal 1(%rax), %edx
```

RISC-V

Отдельная арифметическая команда:

```
addiw a5, a5, 1
```

Windows x86-64

То же, что Linux, но с RCX вместо RDI.

5.4. Передача аргументов (ABI)

ОС/архитектура	Способ передачи аргументов

Linux x86-64	%rdi, %rsi, %rdx, %rcx…
Windows x86-64	%rcx, %rdx, %r8, %r9
RISC-V	a0, a1, a2…

5.5. Особенности RISC-V

- Load/store архитектура → данные читаются только через `lw`/`ld`, записываются через `sw`/`sd`.
- Простой, регулярный набор инструкций.
- Каждая операция работает только с регистрами, без вариантов как `leal`.

---

6. Вывод

В ходе работы изучено:

- Как один и тот же C-код компилируется в разный машинный код.
- Различия ABI между Linux (System V), Windows (MS x64) и RISC-V.
- Как передаются аргументы функций в разных архитектурах.
- Чем load/store архитектуры (RISC-V) отличаются от CISC (x86-64).
- Почему важно понимать ассемблер при системном программировании, написании компиляторов, драйверов и оптимизации кода.