Национальный исследовательский университет “Высшая школа экономики”.

Факультет компьютерных наук. Программная инженерия.

Архитектура вычислительных систем.

Домашнее задание №3 студента группы БПИ213 Абрамова Александра Сергеевича.

Для анализа программ я буду использовать утилиту gdb. Для удобного запуска я создал простой shell-скрипт, который компилирует и линкует программу, название которой задаётся в переменное `NAME`.

```
NAME="01-immediate"

as --gstabs -o $NAME.o $NAME.s

gcc -o $NAME.exe $NAME.o
```

После выполнения указанных команд создаётся файл `$NAME.exe` (где NAME заменено на указанное значение), за поведением которого во время выполнения можно наблюдать с помощью команды

```
gdb ./$NAME.exe
```

В результате анализа было выявлено, что все предложенные программы демонстрируют организацию фрейма функции на стеке при вызове main, а также некоторые другие базовые элементы языка ассемблера:

1. Программа `01-immediate.s`

```
Breakpoint 1, main () at 01-immediate.s:8
8               push    rbp             # save caller's frame pointer
(gdb) i r
rax            0x555555555125      93824992235813
rbx            0x0                 0
rcx            0x7ffff7fba718      140737353852696
rdx            0x7fffffffe5a8      140737488348584
rsi            0x7fffffffe598      140737488348568
rdi            0x1                 1
rbp            0x555555555160      0x555555555160 <__libc_csu_init>
rsp            0x7fffffffe4b8      0x7fffffffe4b8
r8             0x7ffff7fbbd80      140737353858432
r9             0x7ffff7fbbd80      140737353858432
r10            0x0                 0
r11            0x7ffff7f7c188      140737353597320
r12            0x555555555040      93824992235584
r13            0x7fffffffe590      140737488348560
r14            0x0                 0
r15            0x0                 0
rip            0x555555555125      0x555555555125 <main>
eflags         0x246               [ PF ZF IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
```

```
(gdb) step
9               mov     rbp, rsp    # establish our frame pointer
(gdb) step
11              mov     al, 0xab    # 8-bit immediate
(gdb) i r eax
eax            0x55555125          1431654693
(gdb) step
12              mov     ax, 0xabcd  # 16-bit immediate
(gdb) i r eax
eax            0x555551ab          1431654827
(gdb) step
13              mov     eax, 0xabcdef12  # 32-bit immediate
(gdb) i r rax
rax            0x55555555abcd      93824992259021
(gdb) step
14              mov     rax, 0xabcdef12  #    to 64-bit reg
(gdb) i r eax
eax            0xabcdef12          -1412567278
(gdb) i r rax
rax            0xabcdef12          2882400018
(gdb) step
15              mov     rax, 0xabcdef0123456789  # 64-bit immediate
(gdb) i r rax
rax            0xabcdef12          2882400018
(gdb) step
17              mov     eax, 0      # return 0 to os
(gdb) i r rax
rax            0xabcdef0123456789  -6066930334832433271
(gdb) step
18              mov     rsp, rbp    # restore stack pointer
(gdb) i r rax
rax            0x0                 0
(gdb) cont
Continuing.
[Inferior 1 (process 16026) exited normally]
```

Как нетрудно заметить, команда `mov` позволяет записывать некоторые явно заданные значения в регистр `rax` (64 бита) или его младшие части: `al` (8 бит), `ax` (16 бит), `eax` (32 бита). Причём при записи в младшие 32 бита (`eax`), в старшие 32 записываются нули.

2. Программа `02-register.s`

Инструкцию `mov` можно также использовать для копирования значения одного регистра (или его части) в другой регистр такого же размера. Более того, допускается копирование из 64-битного регистра (`r8d`) в 32-битный (`eax`): в таком случае в меньший регистр записывается значение, составленное из первых 32 бит большего регистра.

```
Breakpoint 1, main () at 02-register.s:8
8               push    rbp             # save caller's frame pointer
(gdb) i r
rax            0x555555555125      93824992235813
rbx            0x0                 0
rcx            0x7ffff7fba718      140737353852696
rdx            0x7fffffffe5a8      140737488348584
rsi            0x7fffffffe598      140737488348568
rdi            0x1                 1
rbp            0x555555555150      0x555555555150 <__libc_csu_init>
rsp            0x7fffffffe4b8      0x7fffffffe4b8
r8             0x7ffff7fbbd80      140737353858432
r9             0x7ffff7fbbd80      140737353858432
r10            0x0                 0
r11            0x7ffff7f7c188      140737353597320
r12            0x555555555040      93824992235584
r13            0x7fffffffe590      140737488348560
r14            0x0                 0
r15            0x0                 0
rip            0x555555555125      0x555555555125 <main>
eflags         0x246               [ PF ZF IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
(gdb) step
9               mov     rbp, rsp    # establish our frame pointer
(gdb) step
11              mov     eax, ecx    # 32 bits, low reg codes
(gdb) i r eax ecx
eax            0x55555125          1431654693
ecx            0xf7fba718          -134502632
(gdb) step
12              mov     edi, esi    # highest reg codes
(gdb) i r eax ecx
eax            0xf7fba718          -134502632
ecx            0xf7fba718          -134502632
(gdb) i r edi, esi
Invalid register `edi,'
(gdb) i r edi esi
edi            0x1                 1
esi            0xffffe598          -6760
(gdb) step
13              mov     ax, cx      # 16 bits
(gdb) i r edi esi ax cx
edi            0xffffe598          -6760
esi            0xffffe598          -6760
ax             0xa718              -22760
cx             0xa718              -22760
(gdb) step
14              mov     al, cl      # 8 bits
(gdb) i r ax cx al cl
ax             0xa718              -22760
cx             0xa718              -22760
al             0x18                24
cl             0x18                24
(gdb)
```

```
(gdb) step
15              mov     eax, r8d    # 32 bits, 64-bit register
(gdb) i r al cl eax r8d
al             0x18                24
cl             0x18                24
eax            0xf7fba718          -134502632
r8d            0xf7fbbd80          -134496896
(gdb) step
16              mov     rax, rcx    # 64 bits
(gdb) i r eax r8d rax rcx
eax            0xf7fbbd80          -134496896
r8d            0xf7fbbd80          -134496896
rax            0xf7fbbd80          4160470400
rcx            0x7ffff7fba718      140737353852696
(gdb) step
18              mov     eax, 0      # return 0 to os
(gdb) i r rax, rcx
Invalid register `rax,'
(gdb) i r rax rcx
rax            0x7ffff7fba718      140737353852696
rcx            0x7ffff7fba718      140737353852696
(gdb) cont
Continuing.
[Inferior 1 (process 17849) exited normally]
(gdb)
```

3. Программа `03-memory.s`

```
(gdb) step
9               mov     rbp, rsp    # establish our frame pointer
(gdb) step
10              sub     rsp, 48     # local variables
(gdb) step
12              mov     rcx, 5              # for indexing
(gdb) step
13              mov     eax, [rbp]          # indirect
(gdb) i r eax
eax            0x55555125          1431654693
(gdb) x/lwx $rbp
0x7fffffffe4b0: 0x55555150
(gdb) step
14              mov     eax, -48[rbp]       # indirect + offset
(gdb) i r eax
eax            0x55555150          1431654736
(gdb) x/lwx $rbp-48
0x7fffffffe480: 0xf7fe4530
(gdb) i r eax
eax            0x55555150          1431654736
(gdb) step
15              mov     eax, -48[rbp+rcx]   # indirect + offset and index
(gdb) i r eax
eax            0xf7fe4530          -134331088
(gdb) x/lwx $rbp+rcx-48
No symbol "rcx" in current context.
(gdb) x/lwx $rbp+$rcx-48
0x7fffffffe485: 0x0000007f
```

```
0x7fffffffe485: 0x0000007f
(gdb) step
16              mov     eax, -48[rbp+4*rcx] #       and scaled index
(gdb) i r eax
eax            0x7f                127
(gdb) x/lwx $rbp+4*$rcx-48
0x7fffffffe494: 0x00005555
(gdb) step
18              mov     eax, 0       # return 0 to os
(gdb) i r eax
eax            0x5555              21845
(gdb) cont
Continuing.
[Inferior 1 (process 19628) exited normally]
(gdb)
```

Команда `mov` также позволяет копировать данные, расположенные в памяти по адресу, который хранится в `rbp`, в другие регистры. При этом допускается производить дополнительные вычисления адреса: прибавление явно заданного значения, а также значения регистра `rcx` с некоторым коэффициентом.

4. Программа `04-constToMemory.s`

```
Breakpoint 1, main () at 04-constToMemory.s:8
8               push    rbp         # save caller's frame pointer
(gdb) step
9               mov     rbp, rsp    # establish our frame pointer
(gdb) step
10              sub     rsp, 48     # local variables
(gdb) step
12              mov     rcx, 5              # for indexing
(gdb) step
13              mov     eax, [rbp]          # indirect
(gdb) i r eax
eax            0x55555125          1431654693
(gdb) x/lwx $rbp
0x7fffffffe4a0: 0x55555160
(gdb) step
14              mov     dword ptr -48[rbp], 0x12000034       # indirect + offset
(gdb) i r eax
eax            0x55555160          1431654752
(gdb) x/lwx $rbp-48
0x7fffffffe470: 0xf7fe4530
(gdb) step
15              mov     dword ptr 48[rbp+rcx], 0x56000078   # indirect + offset and index
(gdb) x/lwx $rbp-48
0x7fffffffe470: 0x12000034
(gdb) x/lwx $rbp+$rcx+48
0x7fffffffe4d5: 0xf6000000
(gdb) step
16              mov     dword ptr -48[rbp+4*rcx], 0x91000023 #      and scaled index
(gdb) x/lwx $rbp+$rcp+48
Argument to arithmetic operation not a number or boolean.
(gdb) x/lwx $rbp+$rcx+48
0x7fffffffe4d5: 0x56000078
(gdb) step
18              mov     eax, 0      # return 0 to os
(gdb) x/lwx $rbp+$rcx+48
0x7fffffffe4d5: 0x56000078
(gdb) cont
Continuing.
[Inferior 1 (process 20250) exited normally]
(gdb)
```

С помощью `mov` можно также производить запись в память по определённому адресу, который, аналогично замеченному при анализе программы 3, допускается вычислять.

5. Программа `05-jumps.s`

```
Breakpoint 1, main () at 05-jumps.s:8
8               push    rbp             # save caller's frame pointer
(gdb) step
9               mov     rbp, rsp        # establish our frame pointer
(gdb) step
11              xor     rax, rbx        # sets status flags
(gdb) step
12              jne     forward         # test ZF
(gdb) i r eflags
eflags         0x202               [ IF ]
(gdb) step
forward () at 05-jumps.s:17
17              xor     rax, rbx        # sets status flags
(gdb) i r eflags
eflags         0x202               [ IF ]
(gdb) step
18              je      back            # test ZF
(gdb) i r eflags
eflags         0x202               [ IF ]
(gdb) step
20              mov     eax, 0          # return 0 to os
(gdb) step
21              mov     rsp, rbp        # restore stack pointer
(gdb) step
22              pop     rbp             # restore caller's frame pointer
(gdb) step
forward () at 05-jumps.s:23
23              ret                     # back to caller
(gdb) cont
Continuing.
[Inferior 1 (process 22776) exited normally]
(gdb) q
```

Команды `jne` и `je` позволяют изменять порядок выполнения команд в зависимости от значений флагов регистра `EFLAGS`.

Сравнение некоторых отладчиков

1. GDB

Пользовательский интерфейс: 3

Возможности символьной отладки на уровне исходных текстов: 6

Отображение информации о регистрах: 5

Простота использования: 5

Наличие информации для изучения: 5

Общая субъективная оценка: 5

Демонстрация:

```
(gdb) step
9               mov     rbp, rsp    # establish our frame pointer
(gdb) step
10              sub     rsp, 48     # local variables
(gdb) step
12              mov     rcx, 5              # for indexing
(gdb) step
13              mov     eax, [rbp]          # indirect
(gdb) i r eax
eax            0x55555125          1431654693
(gdb) x/1wx $rbp
0x7fffffffe4b0: 0x55555150
(gdb) step
14              mov     eax, -48[rbp]       # indirect + offset
(gdb) i r eax
eax            0x55555150          1431654736
(gdb) x/1wx $rbp-48
0x7fffffffe480: 0xf7fe4530
(gdb) i r eax
eax            0x55555150          1431654736
(gdb) step
15              mov     eax, -48[rbp+rcx]   # indirect + offset and index
(gdb) i r eax
eax            0xf7fe4530          -134331088
(gdb) x/1wx $rbp+rcx-48
No symbol "rcx" in current context.
(gdb) x/1wx $rbp+$rcx-48
0x7fffffffe485: 0x0000007f
```

2. SASM

Пользовательский интерфейс: 6

Возможности символьной отладки на уровне исходных текстов: 8

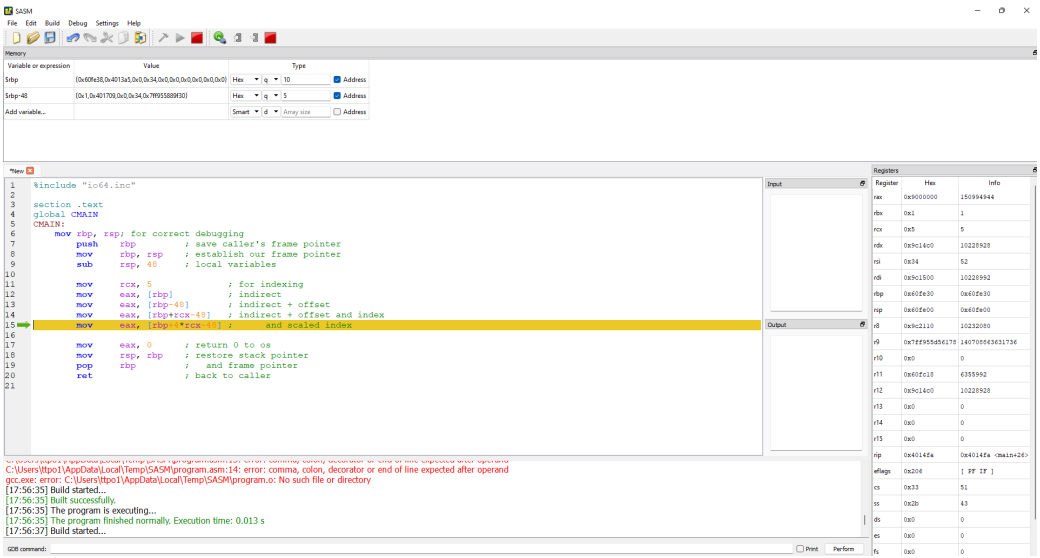
Отображение информации о регистрах: 9

Простота использования: 8

Наличие информации для изучения: 7

Общая субъективная оценка: 8

Демонстрация:



3. MSVS - 2022

Пользовательский интерфейс: 9

Возможности символьной отладки на уровне исходных текстов: 10

Отображение информации о регистрах: 8

Простота использования: 7

Наличие информации для изучения: 8

Общая субъективная оценка: 9

Демонстрация: