



МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра Інформаційної Безпеки

Зворотна розробка та аналіз шкідливого програмного забезпечення

Лабораторна робота №1

Аналіз програмного коду мов високого рівня

Мета:

Отримати навички розпізнавання констукцій мов високого рівня в машинному коді для архітектур x86/x64 та ARM/ARM64 на прикладі C/C++.

Перевірив:

Виконав:

студент II курсу

групи ФБ-01

Сахній Н.Р.

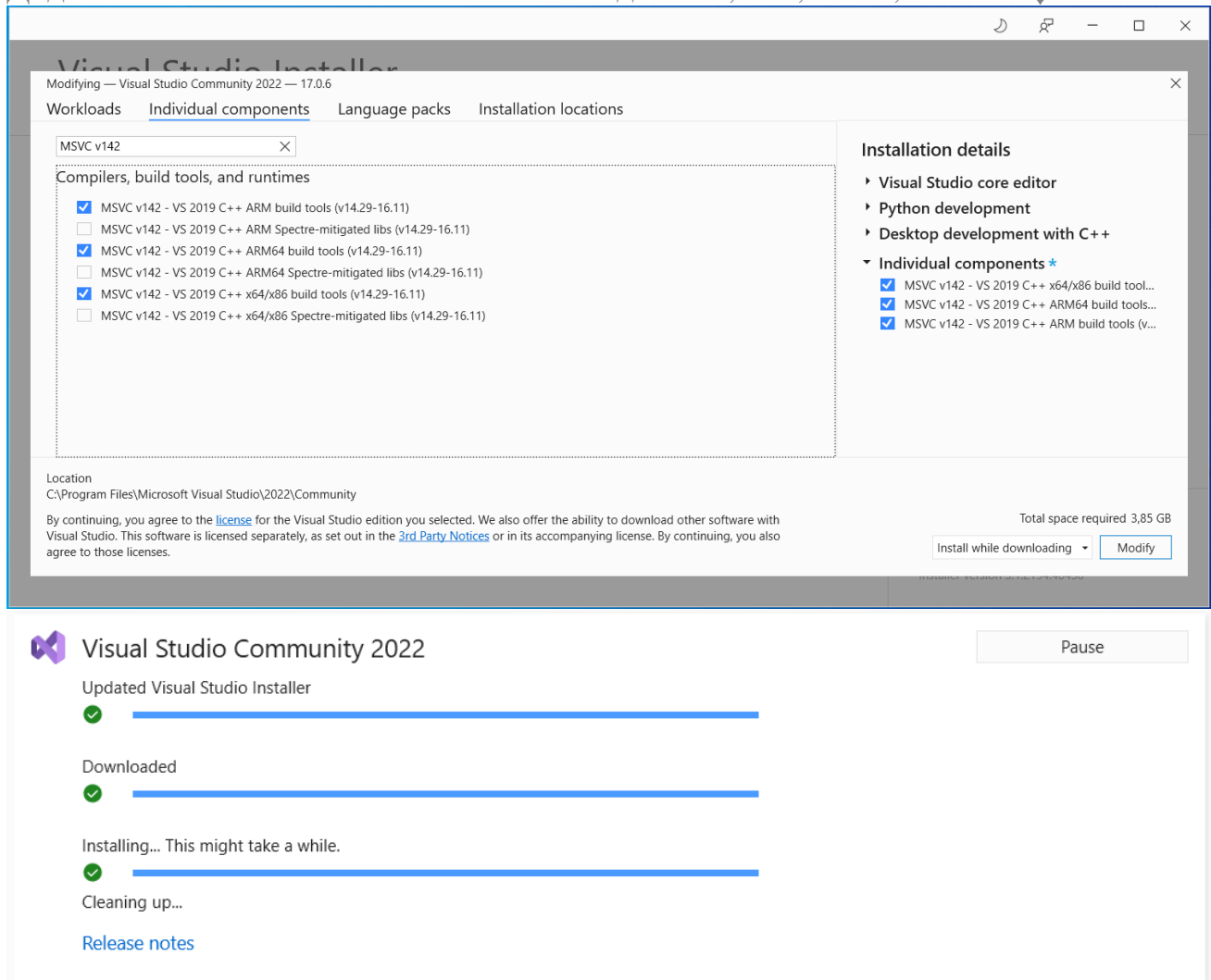
Київ 2022

Завдання до виконання:

Варіант 11

1. Проаналізувати машинний код прикладу hanoi.c для Windows x64, ARM, ARM64 (MSVC), для Linux amd64, arm, arm64 (GCC), для Linux amd64 (LLVM clang, <https://llvm.org/>);

Додатково встановимо пакети build tools для x86, x64, ARM, ARM64 ↓



➤ Microsoft Visual Studio 2022

```
C:\Program Files\Microsoft Visual Studio\2022\Community\VC\Auxiliary\Build>vcvarsall.bat /help
Syntax:
  vcvarsall.bat [arch] [platform_type] [winsdk_version] [-vcvars_ver=vc_version] [-vcvars_spectre_libs=spectre_mode]
where :
  [arch]: x86 | amd64 | x86_amd64 | x86_arm | x86_arm64 | amd64_x86 | amd64_arm | amd64_arm64
  [platform_type]: {empty} | store | uwp
  [winsdk_version] : full Windows 10 SDK number (e.g. 10.0.10240.0) or "8.1" to use the Windows 8.1 SDK.
  [vc_version] : {none} for latest installed VC++ compiler toolset |
                 "14.0" for VC++ 2015 Compiler Toolset |
                 "14.xx" for the latest 14.xx.yyyyy toolset installed (e.g. "14.11") |
                 "14.xx.yyyyy" for a specific full version number (e.g. "14.11.25503")
  [spectre_mode] : {none} for libraries without spectre mitigations |
                 "spectre" for libraries with spectre mitigations
```

Застосувавши компілятор `cl.exe` з параметром `/FA` для кожної цільової архітектури, отримаємо лістинги з асемблерним та машинним кодом, який відповідає окремим рядкам вихідного коду мовою C:

- x64

```
hanoi_x64.cod: Блокнот
Файл Редагування Формат Вигляд Довідка
; 7 : printf("Move disk %d from %c to %c\n", n, from, to);

00058 0f be 44 24 40 movsx    eax, BYTE PTR to$[rsp]
0005d 0f be 4c 24 38 movsx    ecx, BYTE PTR from$[rsp]
00062 44 8b c8      mov     r9d, eax
00065 44 8b c1      mov     r8d, ecx
00068 8b 54 24 30   mov     edx, DWORD PTR n$[rsp]
0006c 48 8d 0d 00 00 00 00 lea     rcx, OFFSET FLAT:$SG9050
00073 e8 00 00 00 00 call    printf
```

- ARM

```
hanoi_ARM.cod: Блокнот
Файл Редагування Формат Вигляд Довідка
; 7 : printf("Move disk %d from %c to %c\n", n, from, to);

00032 f99d 3010     ldrsb    r3, [sp, #0x10]
00036 f99d 200c     ldrsb    r2, [sp, #0xc]
0003a 9902         ldr      r1, [sp, #8]
0003c 4808         ldr      r0, |LN7@hanoi| ; |= $SG5537|
0003e f000 f800     bl      printf
```

- ARM64

```
hanoi_ARM64.cod: Блокнот
Файл Редагування Формат Вигляд Довідка
; 7 : printf("Move disk %d from %c to %c\n", n, from, to);

00074 39c047e3     ldrsb    w3, [sp, #0x11]
00078 2a0303e3     mov     w3, w3
0007c 39c043e2     ldrsb    w2, [sp, #0x10]
00080 2a0203e2     mov     w2, w2
00084 2a0303e3     mov     w3, w3
00088 2a0203e2     mov     w2, w2
0008c b94017e1     ldr      w1, [sp, #0x14]
00090 90000008     adrp    x8, |$SG5285|
00094 91000100     add     x0, x8, |$SG5285|
00098 94000000     bl      printf
```

↑ Аналізуючи для прикладу асемблерний код функції `printf()` (всі лістинги додані в архів), можемо помітити, що головною відмінністю між x64 та ARM/ARM64 є назва регістрів та набір інструкцій для використання ЦП

Бачимо, що відрізняються інструкції переміщення – `movsx` (у x64 перенесення даних меншого розміру у змінну більшого розміру) та `ldrsb` (у ARM/ARM64 перенесення з розширенням знака виконує “Load Register Signed Byte”).

Також можна помітити, що виклик будь-якої функції здійснюється за допомогою різних інструкцій: у x64 це звичний `call`, натомість у arm/arm64 `bl`.

➤ Linux

```
nazar@ubuntu:~/KPI/RevEng$  
nazar@ubuntu:~/KPI/RevEng$ gcc -Wa,-adhln -g ./hanoi.c > hanoi_amd64.lst  
nazar@ubuntu:~/KPI/RevEng$ arm-linux-gnueabi-gcc -Wa,-adhln -g ./hanoi.c > hanoi_arm.lst  
nazar@ubuntu:~/KPI/RevEng$ aarch64-linux-gnu-gcc -Wa,-adhln -g ./hanoi.c > hanoi_arm64.lst  
nazar@ubuntu:~/KPI/RevEng$
```

• amd64

Open		hanoi_amd64.lst	
		~/KPI/RevEng	
63	6:./hanoi.c	****	printf("Move disk %d from %c to %c\n", n, from, to);
64	43	.loc 1 6 3	
65	44 003f 0FBE4DF4	movsbl -12(%rbp), %ecx	
66	45 0043 0FBE55F0	movsbl -16(%rbp), %edx	
67	46 0047 0FBE45F8	movsbl -8(%rbp), %eax	
68	47 004b 8B75FC	movl -4(%rbp), %esi	
69	48 004e 8D7EFF	leal -1(%rsi), %edi	
70	49 0051 89C6	movl %eax, %esi	
71	50 0053 E8000000	call hanoi	
72	50	00	

• arm

Open		hanoi_arm.lst	
		~/KPI/RevEng	
83	6:hanoi.c	****	printf("Move disk %d from %c to %c\n", n, from, to);
84	66	.loc 1 6 3	
85	67 0058 08301BE5	ldr r3, [fp, #-8]	
86	68 005c 010043E2	sub r0, r3, #1	
87	69 0060 0A305BE5	ldrb r3, [fp, #-10] @ zero_extendqisi2	
88	70 0064 0B205BE5	ldrb r2, [fp, #-11] @ zero_extendqisi2	
89	71 0068 09105BE5	ldrb r1, [fp, #-9] @ zero_extendqisi2	
90	72 006c FFFFFFFB	bl hanoi	

• arm64

Open		hanoi_arm64.lst	
		~/KPI/RevEng	
63	6:./hanoi.c	****	printf("Move disk %d from %c to %c\n", n, from, to);
64	45	.loc 1 6 3	
65	46 0044 E01F40B9	ldr w0, [sp, 28]	
66	47 0048 00040051	sub w0, w0, #1	
67	48 004c E36B4039	ldrb w3, [sp, 26]	
68	49 0050 E2674039	ldrb w2, [sp, 25]	
69	50 0054 E16F4039	ldrb w1, [sp, 27]	
70	51 0058 00000094	bl hanoi	

↑ Для аналізу асемблерного коду та порівняння набору компіляторів GNU в Linux з компілятором MSVC cl.exe у Windows було також продемонстровано лістинг фрагменту асемблерного виводу для функції `printf()`. Як можна помітити у GCC більш поширений так званий AT&T синтаксис, проти Intel у MSVS.

2. Реалізувати мовою C/C++, проаналізувати результати компіляції (за варіантом), для платформ i686, amd64, arm, aarch64:

2.1. Комбінаторні алгоритми: (11. на графах - паросполучення)

Приклад програмного коду для даного комбінаторного алгоритму було взято із навчального сайту за наступним посиланням: [Maximum Bipartite Matching](#).

```

nazar@ubuntu:~/KPI/RevEng/Maximum-Bipartite-Matching$ g++ -Wa,-adhln -g Maximum-Bipartite-Matching.cpp > Maximum-Bipartite-Matching_amd64.lst
nazar@ubuntu:~/KPI/RevEng/Maximum-Bipartite-Matching$ i686-linux-gnu-g++ -Wa,-adhln -g Maximum-Bipartite-Matching.cpp > Maximum-Bipartite-Matching_i686.lst
nazar@ubuntu:~/KPI/RevEng/Maximum-Bipartite-Matching$ arm-linux-gnueabi-g++ -Wa,-adhln -g Maximum-Bipartite-Matching.cpp > Maximum-Bipartite-Matching_arm.lst
nazar@ubuntu:~/KPI/RevEng/Maximum-Bipartite-Matching$ aarch64-linux-gnu-g++ -Wa,-adhln -g Maximum-Bipartite-Matching.cpp > Maximum-Bipartite-Matching_arm64.lst
nazar@ubuntu:~/KPI/RevEng/Maximum-Bipartite-Matching$

```

• amd64

```

Maximum-Bipartite-Matching_amd64.lst
~/KPI/RevEng/Maximum-Bipartite-Matching

201 46:Maximum-Bipartite-Matching.cpp **** // Returns maximum number
202 47:Maximum-Bipartite-Matching.cpp **** // of matching from M to N
203 48:Maximum-Bipartite-Matching.cpp **** int maxBPM(bool bpGraph[M][N])
204 49:Maximum-Bipartite-Matching.cpp **** {
205 147 .loc 1 49 1
206 148 .cfi_startproc
207 149 010c F30F1EFA endbr64
208 150 0110 55 pushq %rbp
209 151 .cfi_def_cfa_offset 16
210 152 .cfi_offset 6, -16
211 153 0111 4889E5 movq %rsp, %rbp
212 154 .cfi_def_cfa_register 6
213 155 0114 4883EC40 subq $64, %rsp
214 156 0118 48897DC8 movq %rdi, -56(%rbp)
215 157 .loc 1 49 1
216 158 011c 64488B04 movq %fs:40, %rax
217 158 25280000
218 158 00
219 159 0125 488945F8 movq %rax, -8(%rbp)
220 160 0129 31C0 xorl %eax, %eax

```

• i686

```

Maximum-Bipartite-Matching_i686.lst
~/KPI/RevEng/Maximum-Bipartite-Matching

196 46:Maximum-Bipartite-Matching.cpp **** // Returns maximum number
197 47:Maximum-Bipartite-Matching.cpp **** // of matching from M to N
198 48:Maximum-Bipartite-Matching.cpp **** int maxBPM(bool bpGraph[M][N])
199 49:Maximum-Bipartite-Matching.cpp **** {
200 141 .loc 1 49 1
201 142 .cfi_startproc
202 143 00db F30F1EFB endbr32
203 144 00df 55 pushl %ebp
204 145 .cfi_def_cfa_offset 8
205 146 .cfi_offset 5, -8
206 147 00e0 89E5 movl %esp, %ebp
207 148 .cfi_def_cfa_register 5
208 149 00e2 53 pushl %ebx
209 150 00e3 83EC44 subl $68, %esp
210 151 .cfi_offset 3, -12
211 152 00e6 E8FCFFFF call __x86.get_pc_thunk.bx
212 152 FF
213 153 00eb 81C30200 addl $_GLOBAL_OFFSET_TABLE_, %ebx
214 153 0000
215 154 00f1 8B4508 movl 8(%ebp), %eax
216 155 00f4 8945C4 movl %eax, -60(%ebp)
217 156 .loc 1 49 1
218 157 00f7 65A11400 movl %gs:20, %eax
219 157 0000
220 158 00fd 8945F4 movl %eax, -12(%ebp)
221 159 0100 31C0 xorl %eax, %eax

```

• arm

```

Maximum-Bipartite-Matching_arm.lst
~/KPI/RevEng/Maximum-Bipartite-Matching

223 46:Maximum-Bipartite-Matching.cpp **** // Returns maximum number
224 47:Maximum-Bipartite-Matching.cpp **** // of matching from M to N
225 48:Maximum-Bipartite-Matching.cpp **** int maxBPM(bool bpGraph[M][N])
226 49:Maximum-Bipartite-Matching.cpp **** {
227 182 .loc 1 49 1
228 183 .cfi_startproc
229 184 @ args = 0, pretend = 0, frame = 56
230 185 @ frame_needed = 1, uses_anonymous_args = 0
231 186 0138 00482DE9 push {fp, lr}
232 187 .save {fp, lr}
233 188 .cfi_def_cfa_offset 8
234 189 .cfi_offset 11, -8
235 190 .cfi_offset 14, -4
236 191 .setfp fp, sp, #4
237 192 013c 04B08DE2 add fp, sp, #4
238 193 .cfi_def_cfa 11, 4
239 194 .pad #56
240 195 0140 38D04DE2 sub sp, sp, #56
241 196 0144 38000BE5 str r0, [fp, #-56]
242 197 .loc 1 49 1
243 198 0148 B4309FE5 ldr r3, .L15
244 199 014c 003093E5 ldr r3, [r3]
245 200 0150 08300BE5 str r3, [fp, #-8]
246 201 0154 0030A0E3 mov r3, #0

```

- arm64

```

Maximum-Bipartite-Matching_arm64.lst
~/KPI/RevEng/Maximum-Bipartite-Matching
Save

191 46:Maximum-Bipartite-Matching.cpp **** // Returns maximum number
192 47:Maximum-Bipartite-Matching.cpp **** // of matching from M to N
193 48:Maximum-Bipartite-Matching.cpp **** int maxBPM(bool bpGraph[M][N])
194 49:Maximum-Bipartite-Matching.cpp **** {
195 150 .loc 1 49 1
196 151 .cfi_startproc
197 152 012c FD7BBBA9 stp x29, x30, [sp, -80]!
198 153 .cfi_def_cfa_offset 80
199 154 .cfi_offset 29, -80
200 155 .cfi_offset 30, -72
201 156 0130 FD030091 mov x29, sp
202 157 0134 E00F00F9 str x0, [sp, 24]
203 158 .loc 1 49 1
204 159 0138 00000090 adrp x0, :got:__stack_chk_guard
205 160 013c 000040F9 ldr x0, [x0, #:got_lo12:__stack_chk_guard]
206 161 0140 010040F9 ldr x1, [x0]
207 162 0144 E12700F9 str x1, [sp, 72]
208 163 0148 010080D2 mov x1, 0

```

↑ Щоб продемонструвати, як між собою відрізняються результати компіляції комбінаторного алгоритму, який знаходить максимальну кількість паросполучень в дводольному графі, для платформ amd64, i686, arm, arm64 було взято фрагмент асемблерного коду для функції maxBPM().

2.2. Криптографічні алгоритми, алгоритми кодування та контролю цілісності (11. ChaCha20)

Реалізацію на мові C++ для даного криптографічного алгоритму було взято із репозиторію GitHub за наступним посиланням: [ChaCha20 algorithm](#).

```

nazar@ubuntu:~/KPI/RevEng/ChaCha20$
nazar@ubuntu:~/KPI/RevEng/ChaCha20$ g++ -Wa,-adhln -g chacha20.cpp > ChaCha20_amd64.lst
nazar@ubuntu:~/KPI/RevEng/ChaCha20$ i686-linux-gnu-g++ -Wa,-adhln -g chacha20.cpp > ChaCha20_i686.lst
nazar@ubuntu:~/KPI/RevEng/ChaCha20$ arm-linux-gnueabi-g++ -Wa,-adhln -g chacha20.cpp > ChaCha20_arm.lst
nazar@ubuntu:~/KPI/RevEng/ChaCha20$ aarch64-linux-gnu-g++ -Wa,-adhln -g chacha20.cpp > ChaCha20_arm64.lst
nazar@ubuntu:~/KPI/RevEng/ChaCha20$

```

- amd64

```

*ChaCha20_amd64.lst
~/KPI/RevEng/ChaCha20
Save

1504 121:chacha20.hpp **** Chacha20(
1505 1276 .loc 1 121 5
1506 1277 .cfi_startproc
1507 1278 0000 F30F1EFA endbr64
1508 1279 0004 55 pushq %rbp
1509 1280 .cfi_def_cfa_offset 16
1510 1281 .cfi_offset 6, -16
1511 1282 0005 4889E5 movq %rsp, %rbp
1512 1283 .cfi_def_cfa_register 6
1513 1284 0008 4883EC20 subq $32, %rsp
1514 1285 000c 48897DF8 movq %rdi, -8(%rbp)
1515 1286 0010 488975F0 movq %rsi, -16(%rbp)
1516 1287 0014 488955E8 movq %rdx, -24(%rbp)
1517 1288 0018 48894DE0 movq %rcx, -32(%rbp)
1518 1289 .LBB8:
1519 122:chacha20.hpp **** const uint8_t key[32],
1520 123:chacha20.hpp **** const uint8_t nonce[8],
1521 124:chacha20.hpp **** uint64_t counter = 0

```

- i686

```

Open ChaCha20_i686.lst Save
1592 121:chacha20.hpp **** ChaCha20(
1593 1388 .loc 1 121 5
1594 1389 .cfi_startproc
1595 1390 0000 F30F1EFB endbr32
1596 1391 0004 55 pushl %ebp
1597 1392 .cfi_def_cfa_offset 8
1598 1393 .cfi_offset 5, -8
1599 1394 0005 89E5 movl %esp, %ebp
1600 1395 .cfi_def_cfa_register 5
1601 1396 0007 83EC18 subl $24, %esp
1602 1397 000A E8FCFFFF call __x86.get_pc_thunk.ax
1603 1397 FF
1604 1398 000F 05010000 addl $GLOBAL_OFFSET_TABLE_, %eax
1605 1398 00
1606 1399 0014 8B4514 movl 20(%ebp), %eax
1607 1400 0017 8945F0 movl %eax, -16(%ebp)
1608 1401 001A 8B4518 movl 24(%ebp), %eax
1609 1402 001D 8945F4 movl %eax, -12(%ebp)
1610 1403 .LBB8:
1611 122:chacha20.hpp **** const uint8_t key[32],
1612 123:chacha20.hpp **** const uint8_t nonce[8],
1613 124:chacha20.hpp **** uint64_t counter = 0
1614 125:chacha20.hpp **** ) { block(key, nonce, position(64)) {

```

- arm

```

Open ChaCha20_arm.lst Save
1577 121:chacha20.hpp **** ChaCha20(
1578 1455 .loc 1 121 5
1579 1456 .cfi_startproc
1580 1457 @ args = 8, pretend = 0, frame = 16
1581 1458 @ frame_needed = 1, uses_anonymous_args = 0
1582 1459 0000 00482DE9 push {fp, lr}
1583 1460 .cfi_def_cfa_offset 8
1584 1461 .cfi_offset 11, -8
1585 1462 .cfi_offset 14, -4
1586 1463 0004 04B08DE2 add fp, sp, #4
1587 1464 .cfi_def_cfa 11, 4
1588 1465 0008 10D04DE2 sub sp, sp, #16
1589 1466 000C 08000BE5 str r0, [fp, #-8]
1590 1467 0010 0C100BE5 str r1, [fp, #-12]
1591 1468 0014 10200BE5 str r2, [fp, #-16]
1592 1469 .LBB8:
1593 122:chacha20.hpp **** const uint8_t key[32],
1594 123:chacha20.hpp **** const uint8_t nonce[8],
1595 124:chacha20.hpp **** uint64_t counter = 0
1596 125:chacha20.hpp **** ) { block(key, nonce, position(64)) {

```

- arm64

```

Open ChaCha20_arm64.lst Save
1405 121:chacha20.hpp **** ChaCha20(
1406 1282 .loc 1 121 5
1407 1283 .cfi_startproc
1408 1284 0000 FD7BBDA9 stp x29, x30, [sp, -48]!
1409 1285 .cfi_def_cfa_offset 48
1410 1286 .cfi_offset 29, -48
1411 1287 .cfi_offset 30, -40
1412 1288 0004 FD030091 mov x29, sp
1413 1289 0008 E01700F9 str x0, [sp, 40]
1414 1290 000C E11300F9 str x1, [sp, 32]
1415 1291 0010 E20F00F9 str x2, [sp, 24]
1416 1292 0014 E30B00F9 str x3, [sp, 16]
1417 1293 .LBB8:
1418 122:chacha20.hpp **** const uint8_t key[32],
1419 123:chacha20.hpp **** const uint8_t nonce[8],
1420 124:chacha20.hpp **** uint64_t counter = 0
1421 125:chacha20.hpp **** ) { block(key, nonce, position(64)) {

```

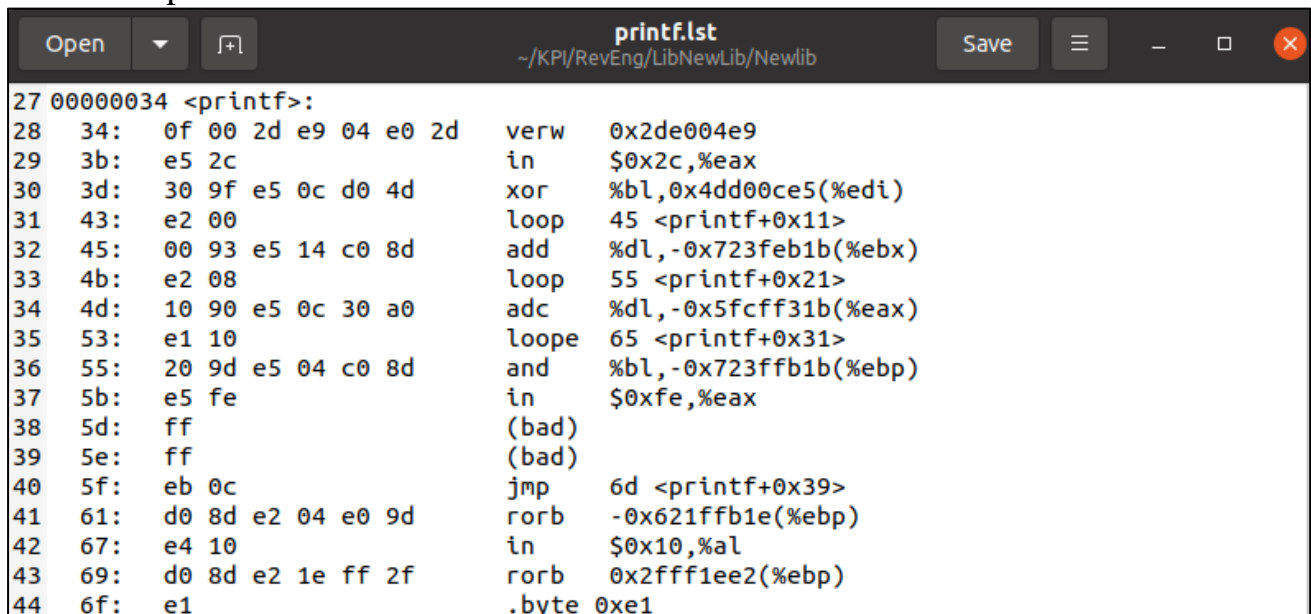
↑ Аби продемонструвати, як між собою відрізняються результати компіляції криптографічного алгоритму ChaCha20 для платформ amd64, i686, arm, arm64 було взято фрагмент асемблерного коду для функції Chacha().

3. Реалізація функцій стандартної бібліотеки C. Бібліотека за варіантами, функції всі зазначені (за наявності реалізації), версія бібліотеки остання стабільна на момент початку курсу: (4. Newlib)

Для того, щоб знайти та завантажити необхідну нам версію бібліотеки необхідно виконати наступні команди ↓

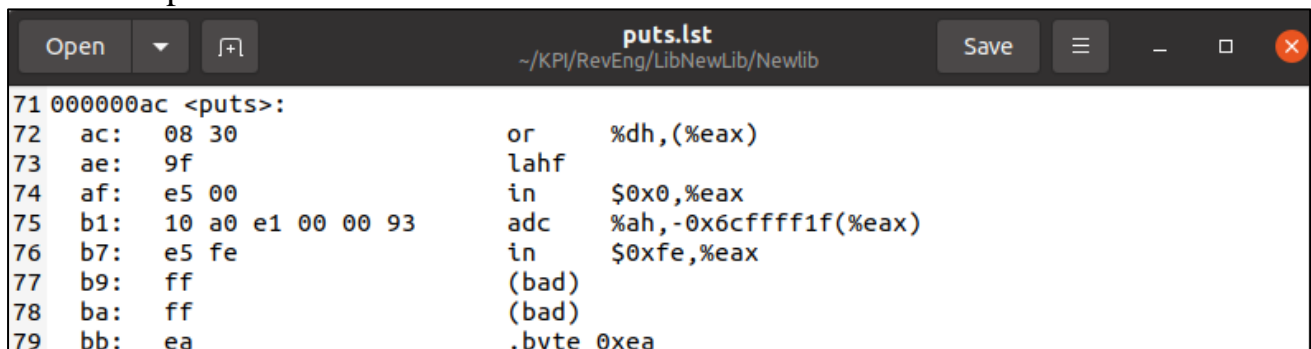
```
nazar@ubuntu:~/KPI/RevEng$  
nazar@ubuntu:~/KPI/RevEng$ apt search libnewlib  
Sorting... Done  
Full Text Search... Done  
libnewlib-arm-none-eabi/focal,focal,now 3.3.0-0ubuntu1 all [installed,automatic]  
  C library and math library compiled for bare metal using Cortex A/R/M  
  
libnewlib-dev/focal,focal,now 3.3.0-0ubuntu1 all [installed,automatic]  
  C library and math library intended for use on embedded systems  
  
libnewlib-doc/focal,focal 3.3.0-0ubuntu1 all  
  C library and math library intended for use on embedded systems (doc)  
  
libnewlib-nano-arm-none-eabi/focal,focal 2.11.2-1 all  
  Smaller embedded C library for arm development  
nazar@ubuntu:~/KPI/RevEng$ apt download libnewlib-arm-none-eabi
```

- Функції стандартного введення-виведення:
 - printf



```
27 00000034 <printf>:  
28 34: 0f 00 2d e9 04 e0 2d   verw    0x2de004e9  
29 3b: e5 2c                   in      $0x2c,%eax  
30 3d: 30 9f e5 0c d0 4d   xor     %bl,0x4dd00ce5(%edi)  
31 43: e2 00                   loop    45 <printf+0x11>  
32 45: 00 93 e5 14 c0 8d   add     %dl,-0x723feb1b(%ebx)  
33 4b: e2 08                   loop    55 <printf+0x21>  
34 4d: 10 90 e5 0c 30 a0   adc     %dl,-0x5fcff31b(%eax)  
35 53: e1 10                   loope   65 <printf+0x31>  
36 55: 20 9d e5 04 c0 8d   and     %bl,-0x723ffb1b(%ebp)  
37 5b: e5 fe                   in      $0xfe,%eax  
38 5d: ff                   (bad)  
39 5e: ff                   (bad)  
40 5f: eb 0c                   jmp     6d <printf+0x39>  
41 61: d0 8d e2 04 e0 9d   rorb    -0x621ffb1e(%ebp)  
42 67: e4 10                   in      $0x10,%al  
43 69: d0 8d e2 1e ff 2f   rorb    0x2fff1ee2(%ebp)  
44 6f: e1                   .byte  0xe1
```

- puts



```
71 000000ac <puts>:  
72 ac: 08 30                   or      %dh,(%eax)  
73 ae: 9f                   lahf  
74 af: e5 00                   in      $0x0,%eax  
75 b1: 10 a0 e1 00 00 93   adc     %ah,-0x6cffff1f(%eax)  
76 b7: e5 fe                   in      $0xfe,%eax  
77 b9: ff                   (bad)  
78 ba: ff                   (bad)  
79 bb: ea                   .byte  0xea
```


- Функції роботи з файлами:
 - fopen

```

Open  ▼  [+]  fopen.lst  Save  [≡]  -  □  ✕
~/KPI/RevEng/LibNewLib/Newlib

98 000000fc <fopen>:
99  fc:  0c 30          or      $0x30,%al
100 fe:  9f            lahf
101 ff:  e5 01          in      $0x1,%eax
102 101: 20 a0 e1 00 10 a0 and     %ah,-0x5fefff1f(%eax)
103 107: e1 00            loope  109 <fopen+0xd>
104 109: 00 93 e5 fe ff ff   add     %dl,-0x11b(%ebx)
105 10f: ea              .byte 0xea

```

- fread

```

Open  ▼  [+]  fread.lst  Save  [≡]  -  □  ✕
~/KPI/RevEng/LibNewLib/Newlib

223 00000274 <fread>:
224 274: 04 e0          add     $0xe0,%al
225 276: 2d e5 24 c0 9f   sub     $0x9fc024e5,%eax
226 27b: e5 0c          in      $0xc,%eax
227 27d: d0 4d e2        rorb    -0x1e(%ebp)
228 280: 00 30          add     %dh,(%eax)
229 282: 8d            (bad)
230 283: e5 02          in      $0x2,%eax
231 285: 30 a0 e1 01 20 a0 xor     %ah,-0x5fdffe1f(%eax)
232 28b: e1 00            loope  28d <fread+0x19>
233 28d: 10 a0 e1 00 00 9c adc     %ah,-0x63ffff1f(%eax)
234 293: e5 fe          in      $0xfe,%eax
235 295: ff            (bad)
236 296: ff            (bad)
237 297: eb 0c          jmp     2a5 <$d+0x1>
238 299: d0 8d e2 04 e0 9d rorb    -0x621ffb1e(%ebp)
239 29f: e4 1e          in      $0x1e,%al
240 2a1: ff 2f          ljmp    *(%edi)
241 2a3: e1              .byte 0xe1

```

- fwrite

```

Open  ▼  [+]  fwrite.lst  Save  [≡]  -  □  ✕
~/KPI/RevEng/LibNewLib/Newlib

71 000000a0 <fwrite>:
72 a0: 04 e0          add     $0xe0,%al
73 a2: 2d e5 24 c0 9f   sub     $0x9fc024e5,%eax
74 a7: e5 0c          in      $0xc,%eax
75 a9: d0 4d e2        rorb    -0x1e(%ebp)
76 ac: 00 30          add     %dh,(%eax)
77 ae: 8d            (bad)
78 af: e5 02          in      $0x2,%eax
79 b1: 30 a0 e1 01 20 a0 xor     %ah,-0x5fdffe1f(%eax)
80 b7: e1 00            loope  b9 <fwrite+0x19>
81 b9: 10 a0 e1 00 00 9c adc     %ah,-0x63ffff1f(%eax)
82 bf: e5 fe          in      $0xfe,%eax
83 c1: ff            (bad)
84 c2: ff            (bad)
85 c3: eb 0c          jmp     d1 <$d+0x1>
86 c5: d0 8d e2 04 e0 9d rorb    -0x621ffb1e(%ebp)
87 cb: e4 1e          in      $0x1e,%al
88 cd: ff 2f          ljmp    *(%edi)
89 cf: e1              .byte 0xe1

```

– feof

```

Open  feof.lst  Save  ~ /KPI/RevEng/LibNewLib/Newlib
7 00000000 <feof>:
8 0: 3c 30      cmp    $0x30,%al
9 2: 9f          lahf
10 3: e5 00      in     $0x0,%eax
11 5: 30 93 e5 00 00 53  xor    %dl,0x530000e5(%ebx)
12 b: e3 10      jecxz  1d <feof+0x1d>
13 d: 40          inc    %eax
14 e: 2d e9 00 40 a0  sub    $0xa04000e9,%eax
15 13: e1 02      loope  17 <feof+0x17>
16 15: 00 00      add    %al,(%eax)
17 17: 0a 38      or     (%eax),%bh
18 19: 20 93 e5 00 00 52  and    %dl,0x520000e5(%ebx)
19 1f: e3 04      jecxz  25 <feof+0x25>
20 21: 00 00      add    %al,(%eax)
21 23: 0a bc 00 d4 e1 a0 02  or     0x2a0e1d4(%eax,%eax,1),%bh
22 2a: a0 e1 01 00 00      mov    0x1e1,%al
23 2f: e2 10      loop   41 <feof+0x41>
24 31: 40          inc    %eax
25 32: bd e8 1e ff 2f      mov    $0x2fff1ee8,%ebp
26 37: e1 03      loope  3c <feof+0x3c>
27 39: 00 a0 e1 fe ff ff      add    %ah,-0x11f(%eax)
28 3f: eb f7      jmp    38 <feof+0x38>
29 41: ff      (bad)
30 42: ff      (bad)
31 43: ea      .byte 0xea

```

– fclose

```

Open  fclose.lst  Save  ~ /KPI/RevEng/LibNewLib/Newlib
101 00000100 <fclose>:
102 100: 08 30      or     %dh,(%eax)
103 102: 9f          lahf
104 103: e5 00      in     $0x0,%eax
105 105: 10 a0 e1 00 00 93  adc    %ah,-0x6cffff1f(%eax)
106 10b: e5 fe      in     $0xfe,%eax
107 10d: ff      (bad)
108 10e: ff      (bad)
109 10f: ea      .byte 0xea

```

- Функції виконання команд операційної системи

– system

```

Open  system.lst  Save  ~ /KPI/RevEng/LibNewLib/Newlib
26 00000030 <system>:
27 30: 00 30      add    %dh,(%eax)
28 32: 50          push   %eax
29 33: e2 01      loop   36 <system+0x6>
30 35: 00 00      add    %al,(%eax)
31 37: 1a 03      sbb    (%ebx),%al
32 39: 00 a0 e1 1e ff 2f  add    %ah,0x2fff1ee1(%eax)
33 3f: e1 10      loope  51 <system+0x21>
34 41: 40          inc    %eax
35 42: 2d e9 fe ff ff  sub    $0xfffffee9,%eax
36 47: eb 58      jmp    a1 <system+0x71>
37 49: 20 a0 e3 00 30 e0  and    %ah,-0x1fcfff1d(%eax)
38 4f: e3 00      jecxz  51 <system+0x21>
39 51: 20 80 e5 10 40 bd  and    %al,-0x42bfef1b(%eax)
40 57: e8 03 00 a0 e1      call   e1a0005f <system+0xe1a0002f>
41 5c: 1e          push   %ds
42 5d: ff 2f      ljmp   *(%edi)
43 5f: e1      .byte 0xe1

```

↑ У наведених вище знімках екрану продемонстровано відповідні фрагменти асемблерного коду кожної з функцій. З їх допомогою можна проаналізувати та зрозуміти, як реалізовано функції стандартної бібліотеки Newlib для мови C.

Висновки:

У цій лабораторній роботі я навчився компілювати програмний код під різні архітектури операційних систем, такі як x86 (i686), x64 (amd64), ARM (arm), ARM64 (aarch64), при цьому використовуючи різні компілятори, а саме: MSVS cl.exe у Windows та набір компіляторів GNU в Linux.

До того ж я отримав навички розпізнавання констукцій мов високого рівня в машинному коді, а також зрозумів які існують відмінності в назвах регістрів та наборі інструкцій для використання ЦП, побудованих на різних архітектурах.