

# REPORT

[Hw 3]



과 목 : 시스템소프트웨어 03

담당교수 : 석문기 교수님

학 과 : 컴퓨터공학과

학 번 : 2021111971

이 름 : 이재혁

# 문제 1

어셈블리의 역할, 오브젝트 파일, 정적 라이브러리가 무엇인지 짧게 설명 하시오.

## 1. 어셈블러

컴파일러에 의해 생성된 어셈블리 언어 코드를 프로세서가 실행할 수 있는 또는 바이너리 코드로 변환하는 역할을 합니다. C 프로그래밍에서 C 소스 코드가 어셈블리 코드로 컴파일 된 후 어셈블러는 이 코드를 처리하여 해당 오브젝트 파일을 생성합니다.

## 2. 오브젝트 파일

C언어에서 오브젝트파일은 어셈블리 언어코드를 어셈블러가 바이너리 코드로 처리한 파일입니다. 기계어 코드가 포함되어 있지만 아직 완전한 실행 파일은 아닙니다. 실행 파일로 결합하는데 사용하는 기호(변수, 함수) 및 재배치 정보와 같은 정보가 포함됩니다.

## 3. 정적 라이브러리

정적 라이브러리는 오브젝트 파일의 집합입니다. C에서 함수 같은 재사용 가능한 코드가 포함된 정적 라이브러리에 포함되어 있습니다. 미리 컴파일 되어있는 정적 라이브러리의 코드를 호출해서 사용하기 때문에 컴파일 실행시간이 줄어들지만, 라이브러리 전체를 포함하고 있어 프로그램의 크기가 커질 수 있습니다.

## 문제 2

오른쪽 코드를 vim을 통해서 직접 입력하시고,  
add.c 를 만들고, 다음의 세가지 캡처본을 만드시오.

gcc -o 옵션을 통해 만들어진 실행 파일의 실행 결과

```
2021111971@linuxserver1:~$ gcc add.c -o add
2021111971@linuxserver1:~$ ./add
Sum : 8
```

gcc -S 를 통해 만들어진 어셈블리

```
GNU nano 4.8                                add.s
.file "add.c"
.text
.globl add
.type add, @function

add:
.LFE0:
.cfi_startproc
endbr64
pushq %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq %rsp, %rbp
.cfi_def_cfa_register 6
movl %edi, -4(%rbp)
movl %esi, -8(%rbp)
movl -4(%rbp), %edx
movl -8(%rbp), %eax
addl %edx, %eax
popq %rbp
.cfi_def_cfa 7, 8
ret
.cfi_endproc
.LFE0:
.size add, .-add
.section .rodata
.LC0:
.string "Sum : %d\n"
.text
.globl main
.type main, @function

main:
.LFB1:
.cfi_startproc
endbr64
pushq %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq %rsp, %rbp
.cfi_def_cfa_register 6
subq $16, %rsp
movl $5, %esi
movl $3, %edi
call add
movl %eax, -4(%rbp)
movl -4(%rbp), %eax
movl %eax, %esi
leaq .LC0(%rip), %rdi
movl $0, %eax
call printf@PLT
movl $0, %eax
leave
.cfi_def_cfa 7, 8
ret
.cfi_endproc
.LFE1:
.size main, .-main
.ident "GCC: (Ubuntu 9.4.0-1ubuntu1-20.04.2) 9.4.0"
.section .note.gnu-stack,"",@progbits
.section .note.gnu.property,"a"
.align 8
.long 1f - 0f
.long 4f - 1f
.long 5
0:
.string "GNU"
1:
.align 8
.long 0xc0000002
.long 3f - 2f
2:
.long 0x3
3:
.align 8
4:
```

# gcc -c 를 통해 만들어진 오브젝트 파일로부터 읽어들이 어셈블리어

- 활용 키워드: `objdump -d`

```
2021111971@kali:~/workspace$ objdump -d add
add:      file format elf64-x86-64

Disassembly of section .init:

0000000000001000 <.init>:
1000: f3 8f 1e fa      endbr64
1004: 48 83 ec 00      sub    $0x4,%esp
1008: 48 8b 85 c0 2f 00 00 mov    0x2f91(Nrip),%rax      # 3f
e8 <_jgcn_start>
100f: 48 85 c0         test   %rax,%rax
1012: 74 02           je     1010 <.init+0x10>
1014: ff 00          callq %rax
1016: 48 83 c4 00     add    $0x4,%rsp
101a: c3            retq

Disassembly of section .plt:

0000000000001020 <.plt>:
1020: ff 35 9a 2f 00 00 pushq  0x2f9a(Nrip)      # 3fc0 <_GLOBAL_OFFSET_TABLE+0x0>
1024: 72 ff 25 9a 2f 00 00 bnd jmq 0x2f9b(Nrip)      # 3fc8 <_GLOBAL_OFFSET_TABLE+0x10>
1028: 0f 2f 00       nopl   (%rax)
102c: f3 8f 1e fa      endbr64
1030: 48 8b 85 c0 2f 00 00 mov    0x2f9b(Nrip),%rax      # 3ff0 <_GLOBAL_OFFSET_TABLE+0x20>
1034: ff 25 9a 2f 00 00 pushq  0x0
1038: f2 49 e1 ff ff ff bnd jmq 1020 <.plt>
103c: 90            nop

Disassembly of section .plt.got:

0000000000001040 <_cxa_finalize@plt>:
1040: f3 8f 1e fa      endbr64
1044: 72 ff 25 ed 2f 00 00 bnd jmq 0x2fad(Nrip)
1048: 0f 2f 44 00 00 nopl   0x0(%rax,%rax,1)

Disassembly of section .plt.sec:
```

```
Disassembly of section .plt.sec:

0000000000001058 <.plt.sec>:
1058: f3 8f 1e fa      endbr64
105c: 72 ff 25 75 2f 00 00 bnd jmq 0x2f76(Nrip)      # 3fd8 <_GLOBAL_OFFSET_TABLE+0x50>
1060: 0f 1f 44 00 00 nopl   0x0(%rax,%rax,1)

Disassembly of section .text:

0000000000001060 <.text>:
1060: f3 8f 1e fa      endbr64
1064: 31 00          xor    %eax,%eax
1068: 49 87 d1        mov    %rcx,%r9
106c: 80           pop    %rsi
1070: 48 89 e1        mov    %rcx,%rcx
1074: 48 8b 84 f0     mov    0x2f90(Nrip),%rcx
1078: 80           pop    %rsi
107c: 4c 8b 96 01 00 00 lea    0x196(Nrip),%rcb      # 1218 <_libc_csu_fini>
1080: 4c 8b 8d 1f 01 00 00 lea    0x11f(Nrip),%rcx      # 11a <_libc_csu_init>
1084: 48 8b 3d 09 00 00 00 lea    0xd9(Nrip),%rdi      # 1161 <main>
1088: ff 10 52 2f 00 00 callq  0x2f52(Nrip)      # 3fe0 <_libc_start@GLIBC_2.2.5>
108c: f4           hit
108e: 90           nop
1090: 90           nop
0000000000001098 <.text>:
1098: 48 8b 3d 79 2f 00 00 lea    0x2f79(Nrip),%rdi      # 48 <_TMC_END_>
109c: 48 8b 85 7f 2f 00 00 lea    0x2f7f(Nrip),%rcx      # 48 <_TMC_END_>
10a0: 48 8b 3f f8     cmp    %rdi,%rcx
10a4: 74 15          je     10b8 <register_tm_clones+0x10>
10a8: 48 8b 85 2a 2f 00 00 mov    0x2f2a(Nrip),%rcx      # 3f <_ITM_deregisterTMCloneTable>
10ac: 48 8b c8       test   %rax,%rax
10b0: 74 00          je     10b8 <register_tm_clones+0x10>
```

```
00000000000010c0 <register_tm_clones>:
10c0: 48 8b 3d 49 2f 00 00 lea    0x2f49(Nrip),%rdi      # 40 <_TMC_END_>
10c4: 48 29 fe      sub    %rdi,%rsi
10c8: 48 89 fe      mov    %rsi,%rcx
10cc: 48 c1 ee 3f    shr    $0x3f,%rsi
10d0: 48 c1 f8 03    sar    $0x3,%rcx
10d4: 48 81 c6      add    %rcx,%rsi
10d8: 48 d1 fe      sar    %rsi
10dc: 74 14          je     10f8 <register_tm_clones+0x38>
10e0: 48 8b 05 05 2f 00 00 mov    0x2f05(Nrip),%rcx      # 3f <_ITM_registerTMCloneTable>
10e4: 48 85 c8       test   %rax,%rax
10e8: 74 00          je     10f8 <register_tm_clones+0x38>
10ec: 74 00          je     10f8 <register_tm_clones+0x38>
10f0: ff e0         jmpq   %rax
10f4: 66 0f 1d 44 00 00 nopw   0x0(%rax,%rax,1)
10f8: c3            retq
10fc: 0f 1f 80 00 00 00 00 nopl   0x0(%rax)
0000000000001100 <_do_global_dtors_aux>:
1100: f3 8f 1e fa      endbr64
1104: 8b 3d 05 2f 00 00 00 cmpb   $0x0,0x2f05(Nrip)      # 40 <_TMC_END_>
1108: 75 2b          jne    1138 <_do_global_dtors_aux+0x38>
110c: 55           push   %rbp
110e: 48 83 3d e2 2e 00 00 cmpq   $0x0,0x2ee2(Nrip)      # 3f <_cxa_finalize@GLIBC_2.2.5>
1110: 00           jmpq   0
1112: 48 89 e5       mov    %rsp,%rbp
1114: 74 6c          je     1127 <_do_global_dtors_aux+0>
```

이하 생략..

## 문제 3

```
(gdb) break main
(gdb) run
(gdb) list
(gdb) disassemble sumstore
(gdb) call add(1,7)
```

커멘트 분석

break main	<pre>(gdb) break main Breakpoint 1 at 0x1161: file add.c, line 9.</pre> <p>add.c 파일의 9 번째 라인의 main 함수에 브레이크 포인트를 설정합니다</p> <p>0x1161 은 main 함수가 실행을 시작하는 메모리의 주소입니다.</p>
run	<pre>(gdb) run Starting program: /home/2021111971/add  Breakpoint 1, main () at add.c:9 9      int main() {</pre> <p>/home/2021111971/add 에 위치한 프로그램을 실행합니다.</p> <p>break main 명령어로 설정한 브레이크 포인트에 중단합니다.</p> <p>9 번째 줄에서 중단되었고, 9 번째 줄을 출력합니다.</p>

list	<pre>(gdb) list 4 5     int add(int a, int b) { 6         return a + b; 7     } 8 9     int main() { 10        int sum = add(3, 5); 11        printf("Sum : %d\n", sum); 12        return 0; 13    }</pre> <p>main 함수를 기준으로 소스코드를 출력합니다</p> <p>10 줄 단위로 출력합니다</p>
disassemble sumstore → disassemble	<pre>(gdb) disassemble sumstore No symbol "sumstore" in current context. (gdb) disassemble Dump of assembler code for function main: =&gt; 0x000055555555161 &lt;+0&gt;:    endbr64 0x000055555555165 &lt;+4&gt;:    push    %rbp 0x000055555555166 &lt;+5&gt;:    mov     %rsp,%rbp 0x000055555555169 &lt;+8&gt;:    sub     \$0x10,%rsp 0x00005555555516d &lt;+12&gt;:   mov     \$0x5,%esi 0x000055555555172 &lt;+17&gt;:   mov     \$0x3,%edi 0x000055555555177 &lt;+22&gt;:   callq   0x55555555149 &lt;add&gt; 0x00005555555517c &lt;+27&gt;:   mov     %eax,-0x4(%rbp) 0x00005555555517f &lt;+30&gt;:   mov     -0x4(%rbp),%eax 0x000055555555182 &lt;+33&gt;:   mov     %eax,%esi 0x000055555555184 &lt;+35&gt;:   lea     0xe79(%rip),%rdi    # 0x555555556004 0x00005555555518b &lt;+42&gt;:   mov     \$0x0,%eax 0x000055555555190 &lt;+47&gt;:   callq   0x55555555050 &lt;printf@plt&gt; 0x000055555555195 &lt;+52&gt;:   mov     \$0x0,%eax 0x00005555555519a &lt;+57&gt;:   leaveq  %eax 0x00005555555519b &lt;+58&gt;:   retq End of assembler dump. (gdb) █</pre> <p>main 함수에서 작동하는 기계 명령어를 표시합니다.</p> <p>시작주소 &lt;사용메모리&gt; 수행 내용으로 출력합니다.</p>
call add(1,7)	<pre>(gdb) call add(1,7) \$1 = 8</pre> <p>1 과 7 을 매개변수로 add 함수를 호출합니다.</p> <p>함수의 반환 값을, \$1 에 저장해 출력합니다.</p>

## 문제 4

### calc.o 파일 생성 후 내용 확인

```
2021111971@linuxserver1:~$ gcc -c calc.c
2021111971@linuxserver1:~$ objdump -d calc.o

calc.o:      file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <add>:
 0:  f3 0f 1e fa          endbr64
 4:  55                   push    %rbp
 5:  48 89 e5             mov     %rsp,%rbp
 8:  89 7d fc             mov     %edi,-0x4(%rbp)
 b:  89 75 f8             mov     %esi,-0x8(%rbp)
 e:  8b 55 fc             mov     -0x4(%rbp),%edx
11:  8b 45 f8             mov     -0x8(%rbp),%eax
14:  01 d0               add     %edx,%eax
16:  5d                   pop     %rbp
17:  c3                   retq

0000000000000018 <minus>:
18:  f3 0f 1e fa          endbr64
1c:  55                   push    %rbp
1d:  48 89 e5             mov     %rsp,%rbp
20:  89 7d fc             mov     %edi,-0x4(%rbp)
23:  89 75 f8             mov     %esi,-0x8(%rbp)
26:  8b 45 fc             mov     -0x4(%rbp),%eax
29:  2b 45 f8             sub     -0x8(%rbp),%eax
2c:  5d                   pop     %rbp
2d:  c3                   retq

000000000000002e <main>:
2e:  f3 0f 1e fa          endbr64
32:  55                   push    %rbp
33:  48 89 e5             mov     %rsp,%rbp
36:  48 83 ec 10          sub     $0x10,%rsp
3a:  be 05 00 00 00       mov     $0x5,%esi
3f:  bf 03 00 00 00       mov     $0x3,%edi
44:  e8 00 00 00 00       callq   49 <main+0x1b>
49:  89 45 f8             mov     %eax,-0x8(%rbp)
4c:  be 05 00 00 00       mov     $0x5,%esi
51:  bf 03 00 00 00       mov     $0x3,%edi
56:  e8 00 00 00 00       callq   5b <main+0x2d>
5b:  89 45 fc             mov     %eax,-0x4(%rbp)
5e:  8b 45 f8             mov     -0x8(%rbp),%eax
61:  89 c6               mov     %eax,%esi
63:  48 8d 3d 00 00 00 00 lea     0x0(%rip),%rdi    # 6a <main+0x3c>
6a:  b8 00 00 00 00       mov     $0x0,%eax
6f:  e8 00 00 00 00       callq   74 <main+0x46>
74:  8b 45 fc             mov     -0x4(%rbp),%eax
77:  89 c6               mov     %eax,%esi
79:  48 8d 3d 00 00 00 00 lea     0x0(%rip),%rdi    # 80 <main+0x52>
80:  b8 00 00 00 00       mov     $0x0,%eax
85:  e8 00 00 00 00       callq   8a <main+0x5c>
8a:  b8 00 00 00 00       mov     $0x0,%eax
8f:  c9                   leaveq  %eax
90:  c3                   retq
```

함수의 하위 주소가 0으로 채워져 있음

### 실행파일 생성 후 내용확인

```
2021111971@linuxserver1:~$ gcc calc.o -o calc
2021111971@linuxserver1:~$ objdump -d calc

calc:      file format elf64-x86-64

Disassembly of section .init:

0000000000001000 <_init>:
1000:  f3 0f 1e fa          endbr64
1004:  48 83 ec 08          sub     $0x8,%rsp
1008:  48 8b 05 d9 2f 00 00 mov     0x2fd9(%rip),%rax    # 3fe8 <__gmon_start__>
100f:  48 85 c0             test    %rax,%rax
1012:  74 02               je      1016 <_init+0x16>
1014:  ff d0               callq   *%rax
1016:  48 83 c4 08          add     $0x8,%rsp
101a:  c3                   retq

Disassembly of section .plt:
```

```

0000000000001177 <main>:
1177:    f3 0f 1e fa    endbr64
117b:    55             push    %rbp
117c:    48 89 e5       mov     %rsp,%rbp
117f:    48 83 ec 10    sub     $0x10,%rsp
1183:    be 05 00 00 00 mov     $0x5,%esi
1188:    bf 03 00 00 00 mov     $0x3,%edi
118d:    e8 b7 ff ff ff callq   1149 <add>
1192:    89 45 f8       mov     %eax,-0x8(%rbp)
1195:    be 05 00 00 00 mov     $0x5,%esi
119a:    bf 03 00 00 00 mov     $0x3,%edi
119f:    e8 bd ff ff ff callq   1161 <minus>
11a4:    89 45 fc       mov     %eax,-0x4(%rbp)
11a7:    8b 45 f8       mov     -0x8(%rbp),%eax
11aa:    89 c6          mov     %eax,%esi
11ac:    48 8d 3d 51 0e 00 00 lea     0xe51(%rip),%rdi    # 2004 <_IO_stdin_
used+0x4>
11b3:    b8 00 00 00 00 mov     $0x0,%eax
11b8:    e8 93 fe ff ff callq   1050 <printf@plt>
11bd:    8b 45 fc       mov     -0x4(%rbp),%eax
11c0:    89 c6          mov     %eax,%esi
11c2:    48 8d 3d 45 0e 00 00 lea     0xe45(%rip),%rdi    # 200e <_IO_stdin_
used+0xe>
11c9:    b8 00 00 00 00 mov     $0x0,%eax
11ce:    e8 7d fe ff ff callq   1050 <printf@plt>
11d3:    b8 00 00 00 00 mov     $0x0,%eax
11d8:    c9            leaveq  %eax
11d9:    c3            retq
11da:    66 0f 1f 44 00 00 nopw    0x0(%rax,%rax,1)

```

함수의 하위비트에 값이 채워졌습니다.

## 링커의 역할

- ➔ 오브젝트 파일에서는 정확한 함수주소가 결정되지 않습니다. 실행파일로 완전히 컴파일 될 때 링커가 함수의 최종 메모리 주소를 결정합니다.