

Nama : Marissa Nafi Aqiela
NIM : 23/516013/PA/22044
Tugas : Assignment 1 Praktikum Sistem Komputer dan Jaringan KOMD
Github : <https://github.com/marissssssa/Marissa-Nafi-Aqiela-SKJ-KOMD>

1.6.2 First Task: C++ Code to Assembly

1. Write a Simple C++ Program

C++ program that adds two integers:

```
#include <iostream>
using namespace std;

int main() {
    int n;

    n = 2 + 3;

    cout << n << endl;

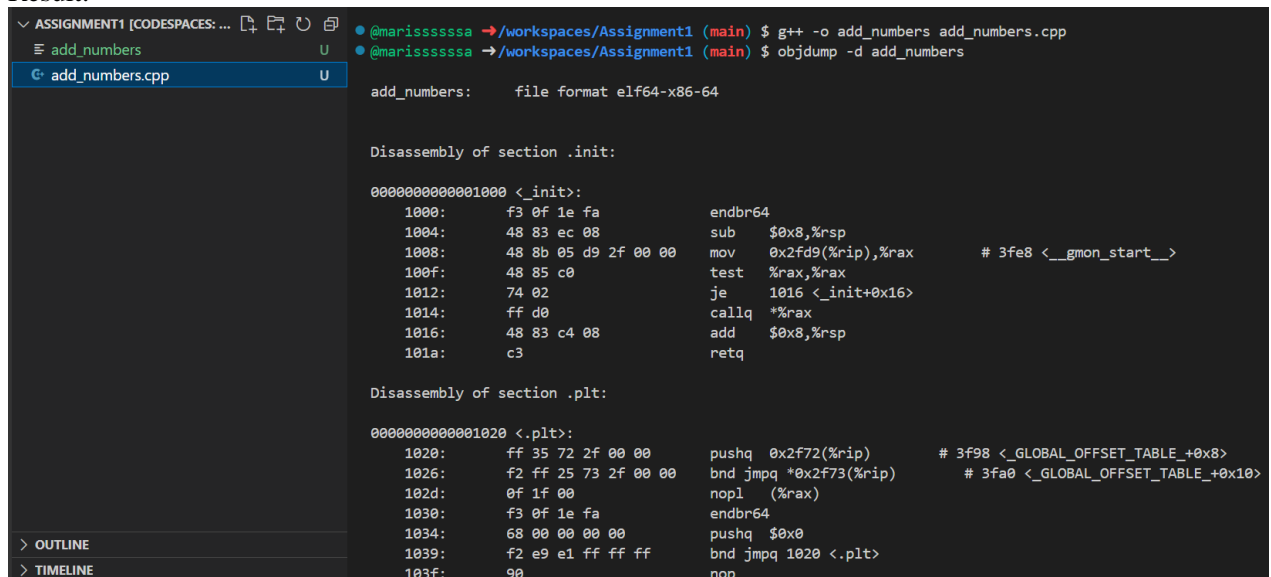
    return 0;
}
```

2. Compile the Code

Use the 'g++' compiler to compile the C++ code. Open a terminal and run the following command:

```
g++ -o add_numbers add_numbers.cpp
```

Result:



```
add_numbers: 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:

0000000000001020 <.plt>:
1020: ff 35 72 2f 00 00 pushq 0x2f72(%rip) # 3f98 <_GLOBAL_OFFSET_TABLE_+0x8>
1026: f2 ff 25 73 2f 00 00 bnd jmpq *0x2f73(%rip) # 3fa0 <_GLOBAL_OFFSET_TABLE_+0x10>
102d: 0f 1f 00        nopl (%rax)
1030: f3 0f 1e fa      endbr64
1034: 68 00 00 00 00 00 pushq $0x0
1039: f2 e9 e1 ff ff ff bnd jmpq 1020<.plt>
103f: 90             nop
```

Gambar ini sudah sekalian dengan hasil disassemble program c++ yang saya buat.

Ketika proses compile ini, program diterjemahkan ke bahasa assembly yang pada step selanjutnya bisa diakses dan dilihat hasil dari bahasa assembly yang sudah di compile. Proses compile (kompilasi) terjadi dengan menerjemahkan kode c++ yang sudah dibuat menjadi kode mesin yang dapat dijalankan oleh prosesor komputer.

3. Disassemble the Code

Disassemble the compiled executable to view the generated assembly code. Use the 'objdump' command as follows:

```
objdump -d add_numbers
```

Hasil disassemble program atau hasil dari penerjemahan c++ ke assembly dari program sebelumnya:

add_numbers: 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:

0000000000001020 <.plt>:

```
1020: ff 35 72 2f 00 00 pushq 0x2f72(%rip)    # 3f98 <_GLOBAL_OFFSET_TABLE_+0x8>
1026: f2 ff 25 73 2f 00 00 bnd jmpq *0x2f73(%rip)    # 3fa0
<_GLOBAL_OFFSET_TABLE_+0x10>
102d: 0f 1f 00      nopl  (%rax)
1030: f3 0f 1e fa    endbr64
1034: 68 00 00 00 00 pushq $0x0
1039: f2 e9 e1 ff ff bnd jmpq 1020 <.plt>
103f: 90            nop
1040: f3 0f 1e fa    endbr64
1044: 68 01 00 00 00 pushq $0x1
1049: f2 e9 d1 ff ff bnd jmpq 1020 <.plt>
104f: 90            nop
1050: f3 0f 1e fa    endbr64
1054: 68 02 00 00 00 pushq $0x2
1059: f2 e9 c1 ff ff bnd jmpq 1020 <.plt>
105f: 90            nop
1060: f3 0f 1e fa    endbr64
1064: 68 03 00 00 00 pushq $0x3
1069: f2 e9 b1 ff ff bnd jmpq 1020 <.plt>
106f: 90            nop
```

Disassembly of section .plt.got:

0000000000001070 <__cxa_finalize@plt>:

```
1070: f3 0f 1e fa    endbr64
1074: f2 ff 25 4d 2f 00 00 bnd jmpq *0x2f4d(%rip)    # 3fc8 <__cxa_finalize@GLIBC_2.2.5>
107b: 0f 1f 44 00 00 nopl  0x0(%rax,%rax,1)
```

Disassembly of section .plt.sec:

0000000000001080 <__cxa_atexit@plt>:

```
1080: f3 0f 1e fa    endbr64
1084: f2 ff 25 1d 2f 00 00 bnd jmpq *0x2f1d(%rip)    # 3fa8 <__cxa_atexit@GLIBC_2.2.5>
108b: 0f 1f 44 00 00 nopl  0x0(%rax,%rax,1)
```

0000000000001090 <_ZNSolsEPFRSoS_E@plt>:

```
1090: f3 0f 1e fa    endbr64
1094: f2 ff 25 15 2f 00 00 bnd jmpq *0x2f15(%rip)    # 3fb0
```

```

<_ZNSolsEPFRSoS_E@GLIBCXX_3.4>
109b: 0f 1f 44 00 00    nopl 0x0(%rax,%rax,1)

00000000000010a0 <_ZNSt8ios_base4InitC1Ev@plt>:
10a0: f3 0f 1e fa      endbr64
10a4: f2 ff 25 0d 2f 00 00    bnd jmpq *0x2f0d(%rip)    # 3fb8
<_ZNSt8ios_base4InitC1Ev@GLIBCXX_3.4>
10ab: 0f 1f 44 00 00    nopl 0x0(%rax,%rax,1)

00000000000010b0 <_ZNSolsEi@plt>:
10b0: f3 0f 1e fa      endbr64
10b4: f2 ff 25 05 2f 00 00    bnd jmpq *0x2f05(%rip)    # 3fc0 <_ZNSolsEi@GLIBCXX_3.4>
10bb: 0f 1f 44 00 00    nopl 0x0(%rax,%rax,1)

Disassembly of section .text:

00000000000010c0 <_start>:
10c0: f3 0f 1e fa      endbr64
10c4: 31 ed            xor %ebp,%ebp
10c6: 49 89 d1          mov %rdx,%r9
10c9: 5e                pop %rsi
10ca: 48 89 e2          mov %rsp,%rdx
10cd: 48 83 e4 f0       and $0xfffffffffffff0,%rsp
10d1: 50                push %rax
10d2: 54                push %rsp
10d3: 4c 8d 05 e6 01 00 00    lea 0x1e6(%rip),%r8    # 12c0 <__libc_csu_fini>
10da: 48 8d 0d 6f 01 00 00    lea 0x16f(%rip),%rcx    # 1250 <__libc_csu_init>
10e1: 48 8d 3d c1 00 00 00    lea 0xc1(%rip),%rdi    # 11a9 <main>
10e8: ff 15 f2 2e 00 00    callq *0x2ef2(%rip)    # 3fe0 <__libc_start_main@GLIBC_2.2.5>
10ee: f4                hlt
10ef: 90                nop

00000000000010f0 <deregister_tm_clones>:
10f0: 48 8d 3d 19 2f 00 00    lea 0x2f19(%rip),%rdi    # 4010 <__TMC_END__>
10f7: 48 8d 05 12 2f 00 00    lea 0x2f12(%rip),%rax    # 4010 <__TMC_END__>
10fe: 48 39 f8          cmp %rdi,%rax
1101: 74 15            je 1118 <deregister_tm_clones+0x28>
1103: 48 8b 05 ce 2e 00 00    mov 0x2ece(%rip),%rax    # 3fd8 <_ITM_deregisterTMCloneTable>
110a: 48 85 c0          test %rax,%rax
110d: 74 09            je 1118 <deregister_tm_clones+0x28>
110f: ff e0            jmpq *%rax
1111: 0f 1f 80 00 00 00 00    nopl 0x0(%rax)
1118: c3                retq
1119: 0f 1f 80 00 00 00 00    nopl 0x0(%rax)

0000000000001120 <register_tm_clones>:
1120: 48 8d 3d e9 2e 00 00    lea 0x2ee9(%rip),%rdi    # 4010 <__TMC_END__>
1127: 48 8d 35 e2 2e 00 00    lea 0x2ee2(%rip),%rsi    # 4010 <__TMC_END__>
112e: 48 29 fe          sub %rdi,%rsi
1131: 48 89 f0          mov %rsi,%rax
1134: 48 c1 ee 3f       shr $0x3f,%rsi
1138: 48 c1 f8 03       sar $0x3,%rax
113c: 48 01 c6          add %rax,%rsi
113f: 48 d1 fe          sar %rsi
1142: 74 14            je 1158 <register_tm_clones+0x38>
1144: 48 8b 05 a5 2e 00 00    mov 0x2ea5(%rip),%rax    # 3ff0 <_ITM_registerTMCloneTable>
114b: 48 85 c0          test %rax,%rax
114e: 74 08            je 1158 <register_tm_clones+0x38>
1150: ff e0            jmpq *%rax
1152: 66 0f 1f 44 00 00    nopw 0x0(%rax,%rax,1)
1158: c3                retq
1159: 0f 1f 80 00 00 00 00    nopl 0x0(%rax)

0000000000001160 <__do_global_dtors_aux>:
1160: f3 0f 1e fa      endbr64

```

```

1164: 80 3d e5 2f 00 00 00  cmpb $0x0,0x2fe5(%rip)    # 4150 <completed.8061>
116b: 75 2b                jne 1198 <__do_global_dtors_aux+0x38>
116d: 55                  push %rbp
116e: 48 83 3d 52 2e 00 00  cmpq $0x0,0x2e52(%rip)    # 3fc8 <__cxa_finalize@GLIBC_2.2.5>
1175: 00
1176: 48 89 e5            mov  %rsp,%rbp
1179: 74 0c                je 1187 <__do_global_dtors_aux+0x27>
117b: 48 8b 3d 86 2e 00 00  mov  0x2e86(%rip),%rdi    # 4008 <__dso_handle>
1182: e8 e9 fe ff ff      callq 1070 <__cxa_finalize@plt>
1187: e8 64 ff ff ff      callq 10f0 <deregister_tm_clones>
118c: c6 05 bd 2f 00 00 01  movb $0x1,0x2fbd(%rip)    # 4150 <completed.8061>
1193: 5d                  pop  %rbp
1194: c3                  retq
1195: 0f 1f 00            nopl (%rax)
1198: c3                  retq
1199: 0f 1f 80 00 00 00 00  nopl 0x0(%rax)

```

00000000000011a0 <frame_dummy>:

```

11a0: f3 0f 1e fa        endbr64
11a4: e9 77 ff ff ff      jmpq 1120 <register_tm_clones>

```

00000000000011a9 <main>:

```

11a9: f3 0f 1e fa        endbr64
11ad: 55                  push %rbp
11ae: 48 89 e5            mov  %rsp,%rbp
11b1: 48 83 ec 10         sub  $0x10,%rsp
11b5: c7 45 fc 0a 00 00 00  movl $0xa,-0x4(%rbp)
11bc: 8b 45 fc            mov  -0x4(%rbp),%eax
11bf: 89 c6              mov  %eax,%esi
11c1: 48 8d 3d 78 2e 00 00  lea  0x2e78(%rip),%rdi    # 4040 <_ZSt4cout@@GLIBCXX_3.4>
11c8: e8 e3 fe ff ff      callq 10b0 <_ZNSolsEi@plt>
11cd: 48 89 c2            mov  %rax,%rdx
11d0: 48 8b 05 f9 2d 00 00  mov  0x2df9(%rip),%rax    # 3fd0
<_ZSt4endlcSt11char_traits1cEERSt13basic_ostreamIT_T0_ES6_@GLIBCXX_3.4>
11d7: 48 89 c6            mov  %rax,%rsi
11da: 48 89 d7            mov  %rdx,%rdi
11dd: e8 ae fe ff ff      callq 1090 <_ZNSolsEPFRSoS_E@plt>
11e2: b8 00 00 00 00      mov  $0x0,%eax
11e7: c9                  leaveq
11e8: c3                  retq

```

00000000000011e9 <_Z41__static_initialization_and_destruction_0ii>:

```

11e9: f3 0f 1e fa        endbr64
11ed: 55                  push %rbp
11ee: 48 89 e5            mov  %rsp,%rbp
11f1: 48 83 ec 10         sub  $0x10,%rsp
11f5: 89 7d fc            mov  %edi,-0x4(%rbp)
11f8: 89 75 f8            mov  %esi,-0x8(%rbp)
11fb: 83 7d fc 01         cmpl $0x1,-0x4(%rbp)
11ff: 75 32                jne 1233 <_Z41__static_initialization_and_destruction_0ii+0x4a>
1201: 81 7d f8 ff ff 00 00  cmpl $0xffff,-0x8(%rbp)
1208: 75 29                jne 1233 <_Z41__static_initialization_and_destruction_0ii+0x4a>
120a: 48 8d 3d 40 2f 00 00  lea  0x2f40(%rip),%rdi    # 4151 <_ZStL8__ioinit>
1211: e8 8a fe ff ff      callq 10a0 <_ZNSt8ios_base4InitC1Ev@plt>
1216: 48 8d 15 eb 2d 00 00  lea  0x2deb(%rip),%rdx    # 4008 <__dso_handle>
121d: 48 8d 35 2d 2f 00 00  lea  0x2f2d(%rip),%rsi    # 4151 <_ZStL8__ioinit>
1224: 48 8b 05 cd 2d 00 00  mov  0x2dcd(%rip),%rax    # 3ff8
<_ZNSt8ios_base4InitD1Ev@GLIBCXX_3.4>
122b: 48 89 c7            mov  %rax,%rdi
122e: e8 4d fe ff ff      callq 1080 <__cxa_atexit@plt>
1233: 90                  nop
1234: c9                  leaveq
1235: c3                  retq

```

0000000000001236 <_GLOBAL__sub_I_main>:

```

1236: f3 0f 1e fa    endbr64
123a: 55            push %rbp
123b: 48 89 e5      mov  %rsp,%rbp
123e: be ff ff 00 00 mov  $0xffff,%esi
1243: bf 01 00 00 00 mov  $0x1,%edi
1248: e8 9c ff ff ff callq 11e9 <_Z41__static_initialization_and_destruction_0ii>
124d: 5d            pop  %rbp
124e: c3            retq
124f: 90            nop

```

0000000000001250 <__libc_csu_init>:

```

1250: f3 0f 1e fa    endbr64
1254: 41 57          push %r15
1256: 4c 8d 3d 1b 2b 00 00 lea  0x2b1b(%rip),%r15    # 3d78

```

<__frame_dummy_init_array_entry>

```

125d: 41 56          push %r14
125f: 49 89 d6      mov  %rdx,%r14
1262: 41 55          push %r13
1264: 49 89 f5      mov  %rsi,%r13
1267: 41 54          push %r12
1269: 41 89 fc      mov  %edi,%r12d
126c: 55            push %rbp
126d: 48 8d 2d 14 2b 00 00 lea  0x2b14(%rip),%rbp    # 3d88

```

<__do_global_dtors_aux_fini_array_entry>

```

1274: 53            push %rbx
1275: 4c 29 fd      sub  %r15,%rbp
1278: 48 83 ec 08   sub  $0x8,%rsp
127c: e8 7f fd ff ff callq 1000 <_init>
1281: 48 c1 fd 03   sar  $0x3,%rbp
1285: 74 1f        je   12a6 <__libc_csu_init+0x56>
1287: 31 db        xor  %ebx,%ebx
1289: 0f 1f 80 00 00 00 00 nopl 0x0(%rax)
1290: 4c 89 f2      mov  %r14,%rdx
1293: 4c 89 ee      mov  %r13,%rsi
1296: 44 89 e7      mov  %r12d,%edi
1299: 41 ff 14 df   callq *(%r15,%rbx,8)
129d: 48 83 c3 01   add  $0x1,%rbx
12a1: 48 39 dd      cmp  %rbx,%rbp
12a4: 75 ea        jne  1290 <__libc_csu_init+0x40>
12a6: 48 83 c4 08   add  $0x8,%rsp
12aa: 5b            pop  %rbx
12ab: 5d            pop  %rbp
12ac: 41 5c        pop  %r12
12ae: 41 5d        pop  %r13
12b0: 41 5e        pop  %r14
12b2: 41 5f        pop  %r15
12b4: c3            retq
12b5: 66 66 2e 0f 1f 84 00 data16 nopw %cs:0x0(%rax,%rax,1)
12bc: 00 00 00 00

```

00000000000012c0 <__libc_csu_fini>:

```

12c0: f3 0f 1e fa    endbr64
12c4: c3            retq

```

Disassembly of section .fini:

00000000000012c8 <_fini>:

```

12c8: f3 0f 1e fa    endbr64
12cc: 48 83 ec 08   sub  $0x8,%rsp
12d0: 48 83 c4 08   add  $0x8,%rsp
12d4: c3            retq

```

Objdump command yang digunakan adalah alat yang digunakan untuk disassemble program, yang berarti ia memecah kode mesin biner dari file eksekusi menjadi instruksi assembly yang lebih mudah dipahami oleh manusia. Ini seperti penerjemah yang mengambil kode yang telah dikompilasi oleh compiler (dalam bentuk biner yang hanya dipahami oleh mesin) dan mengubahnya kembali menjadi bahasa assembly, yang merupakan representasi lebih mendasar dari apa yang dilakukan oleh CPU. Dengan disassembly ini, kita bisa melihat langkah-langkah yang diambil oleh prosesor saat menjalankan program.

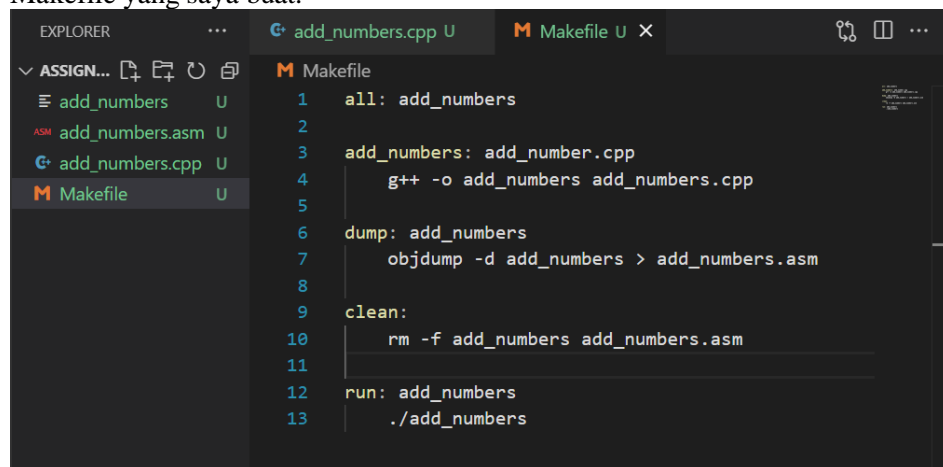
Pada program c++ yang dibuat sebelumnya, yang terlihat sangat sederhana, namun menghasilkan kode assembly yang sangat panjang. Ini terjadi karena program c++ masih harus menginisialisasi iostream, mengatur input/output, mengelola memori, dan melakukan optimasi agar program berjalan efisien. Selain itu, ada juga penambahan kode untuk menangani keamanan dan error. Ini semua memerlukan sejumlah besar instruksi assembly yang disebabkan oleh kompleksitas library pada C++, membuat program yang tampak sederhana dalam kode sumber menjadi jauh lebih kompleks dalam bentuk bahasa assembly.

4. Write a Makefile

Create a 'Makefile' that includes the following targets:

- 'all': Compiles the code.
- 'dump': Disassembles the compiled code.
- 'clean': Removes all output files created by 'make all' and 'make dump', but not the source code.
- 'run': Runs the compiled executable.

Makefile yang saya buat:



```
1  all: add_numbers
2
3  add_numbers: add_number.cpp
4      g++ -o add_numbers add_numbers.cpp
5
6  dump: add_numbers
7      objdump -d add_numbers > add_numbers.asm
8
9  clean:
10     rm -f add_numbers add_numbers.asm
11
12  run: add_numbers
13     ./add_numbers
```

Hasil mencoba disassemble kode menggunakan command `make dump` yang mana akan disassemble file executable `add_numbers` dan menyimpan hasilnya ke dalam file `add_numbers.asm`. Dan hasilnya terlihat pada gambar dibawah ini:

```

1  add_numbers: file format mach-o arm64
2
3
4  Disassembly of section __TEXT,__text:
5
6  0000000100003d40 <_main>:
7  100003d40: ff 83 00 d1 sub sp, sp, #32
8  100003d44: fd 7b 01 a9 stp x29, x30, [sp, #16]
9  100003d48: fd 43 00 91 add x29, sp, #16
10 100003d4c: 08 00 00 52 mov w8, #0
11 100003d50: e8 07 00 b9 str w8, [sp, #4]
12 100003d54: bf c3 1f b8 stur wzr, [x29, #-4]
13 100003d58: a8 00 00 52 mov w8, #5
14 100003d5c: e8 0b 00 b9 str w8, [sp, #8]
15 100003d60: e1 0b 40 b9 ldr w1, [sp, #8]
16 100003d64: 00 00 00 b0 adrp x0, 0x100004000 <_main+0x28>
17 100003d68: 00 18 40 f9 ldr x0, [x0, #48]
18 100003d6c: 77 00 00 94 bl 0x100003f48 <__gxx_personality_v0+0x100003f48>
19 100003d70: 01 00 00 b0 adrp x1, 0x100004000 <_main+0x34>
20 100003d74: 21 1c 40 f9 ldr x1, [x1, #56]
21 100003d78: 05 00 00 94 bl 0x100003d8c <_ZNSt3__13basic_ostreamInS_11char_traitsIcEEEElsB6v15006EPFRS3_S4_E>
22 100003d7c: e0 07 40 b9 ldr w0, [sp, #4]
23 100003d80: fd 7b 41 a9 ldp x29, x30, [sp, #16]
24 100003d84: ff 83 00 91 add sp, sp, #32
25 100003d88: c0 03 5f d6 ret
26
27 0000000100003d8c <_ZNSt3__13basic_ostreamInS_11char_traitsIcEEEElsB6v15006EPFRS3_S4_E>:
28 100003d8c: ff 83 00 d1 sub sp, sp, #32
29 100003d90: fd 7b 01 a9 stp x29, x30, [sp, #16]
30 100003d94: fd 43 00 91 add x29, sp, #16
31 100003d98: e0 07 00 f9 str x0, [sp, #8]
32 100003d9c: e1 03 00 f9 str x1, [sp]

```

Menggunakan command `make clean` yang mana membersihkan file hasil kompilasi dan disassembly. Dan hasilnya file `add_numbers` dan `add_numbers.asm` sudah terhapus.

```

1  all: add_numbers
2
3  add_numbers: add_numbers.cpp
4  g++ -o add_numbers add_numbers.cpp
5
6  dump: add_numbers
7  objdump -d add_numbers > add_numbers.asm
8
9  clean:
10 rm -f add_numbers add_numbers.asm
11
12 run: add_numbers
13 ./add_numbers
14

```

PROBLEMS	OUTPUT	DEBUG CONSOLE	TERMINAL	PORTS
100003f30:	10 00 00 b0	adrp	x16, 0x100004000 <__stubs+0x28>	
100003f34:	10 0e 40 f9	ldr	x16, [x16, #24]	
100003f38:	00 02 1f d6	br	x16	
100003f3c:	10 00 00 b0	adrp	x16, 0x100004000 <__stubs+0x34>	
100003f40:	10 12 40 f9	ldr	x16, [x16, #32]	
100003f44:	00 02 1f d6	br	x16	
100003f48:	10 00 00 b0	adrp	x16, 0x100004000 <__stubs+0x40>	
100003f4c:	10 16 40 f9	ldr	x16, [x16, #40]	
100003f50:	00 02 1f d6	br	x16	
100003f54:	10 00 00 b0	adrp	x16, 0x100004000 <__stubs+0x4c>	
100003f58:	10 26 40 f9	ldr	x16, [x16, #72]	
100003f5c:	00 02 1f d6	br	x16	
100003f60:	10 00 00 b0	adrp	x16, 0x100004000 <__stubs+0x58>	
100003f64:	10 2a 40 f9	ldr	x16, [x16, #80]	
100003f68:	00 02 1f d6	br	x16	
100003f6c:	10 00 00 b0	adrp	x16, 0x100004000 <__stubs+0x64>	
100003f70:	10 2e 40 f9	ldr	x16, [x16, #88]	
100003f74:	00 02 1f d6	br	x16	

Second Task: Assembly to C++

5. Analyze the Provided Assembly Code

Consider the following assembly code (for illustration purposes; it may not compile directly):

```
section .data
    num1 dw 5
    num2 dw 10
    result dw 0
section .text
    global _start
_start:
    mov ax, [num1]
    imul ax, [num2]
    mov [result], ax
    ; Exit the program
    mov eax, 1
    xor ebx, ebx
    int 0x80
```

Explain and describe each line of the code.

Jawab:

section .data : Ini adalah bagian deklarasi data dalam program. Di sini kita mendefinisikan variabel-variabel yang akan digunakan.

num1 dw 5 : Baris ini mendeklarasikan variabel bernama 'num1' dan memberikan nilai 5. 'dw' berarti define word, yang mengalokasikan ruang memori 16-bit untuk menyimpan nilai tersebut.

num2 dw 10 : Serupa dengan baris sebelumnya, ini mendeklarasikan variabel 'num2' dan memberikan nilai 10.

result dw 0 : Ini mendeklarasikan variabel 'result' untuk menyimpan hasil perhitungan nantinya. Saat ini diinisialisasi dengan nilai 0.

section .text : Bagian ini menandai awal dari segmen kode yang akan dieksekusi.

global _start : Baris ini mendeklarasikan simbol '_start' sebagai global, yang berarti dapat diakses oleh linker.

_start: Ini adalah label yang menandai titik awal eksekusi program.

mov ax, [num1] : Instruksi ini memindahkan nilai dari alamat memori 'num1' ke register AX.

imul ax, [num2] : Ini melakukan perkalian integer antara nilai di register AX dengan nilai di alamat memori 'num2'. Hasilnya disimpan kembali di AX.

mov [result], ax : Instruksi ini memindahkan hasil perkalian dari register AX ke alamat memori 'result'.

; Exit the program : Ini adalah komentar yang menjelaskan tujuan kode berikutnya.

mov eax, 1 : Mempersiapkan sistem untuk keluar dari program dengan memindahkan nilai 1 ke register EAX.

xor ebx, ebx : Menggunakan operasi XOR untuk menset register EBX ke 0, yang menandakan program berakhir tanpa error.

int 0x80 : Ini adalah interrupt yang memanggil sistem operasi untuk mengakhiri program.

Pada dasarnya, program ini bermaksud untuk mengambil dua nilai (5 dan 10), mengalikannya, menyimpan hasilnya, lalu mengakhiri eksekusi.

Write the Equivalent C++ Code

Based on the provided assembly code, write a C++ program that performs the same functionality. The C++ program should produce the same result as the assembly code.

```
#include <iostream> using
namespace std;

int main() {
    // Equivalent to the 'section .data' in assembly short
    num1 = 5;
    short num2 = 10; short result
    = 0;

    // Equivalent to the computation in the 'section .text' result = num1
    * num2;

    cout << "Result: " << result << endl;

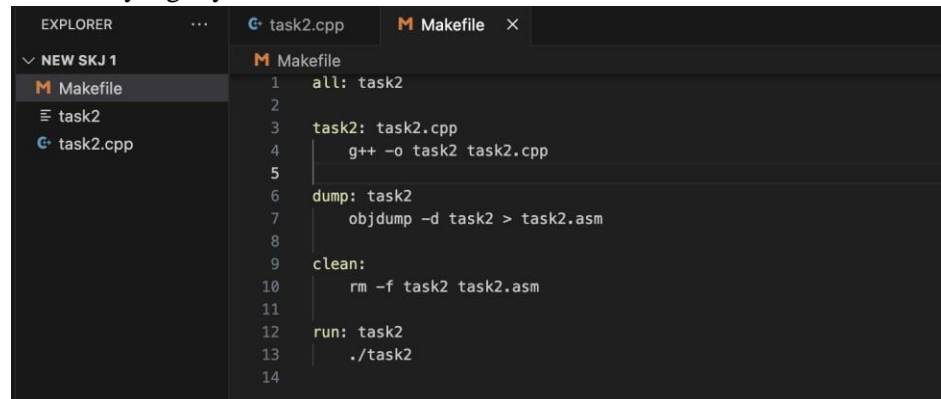
    // The program will automatically exit here, equivalent to the exit code in
    assembly
```

Program C++ yang saya buat serupa dengan kode assembly yang diberikan pada soal, karena keduanya mengikuti struktur dasar yang sama dalam hal pengelolaan data dan instruksi eksekusi. Pada program C++, variabel `num1`, `num2`, dan `result` berfungsi seperti bagian `.data` dalam assembly, tempat data disimpan. Operasi `result = num1 * num2;` dalam C++ setara dengan instruksi `mov` dan `imul` dalam assembly, di mana nilai dari `num1` dan `num2` diambil, dikalikan, dan hasilnya disimpan di `result`. Akhirnya, perintah `cout` di C++ mengeluarkan hasilnya dan program berakhir dengan `return 0;;`, yang mirip dengan kode assembly yang menggunakan `mov`, `xor`, dan `int 0x80` untuk mengakhiri program.

3. Write a Makefile

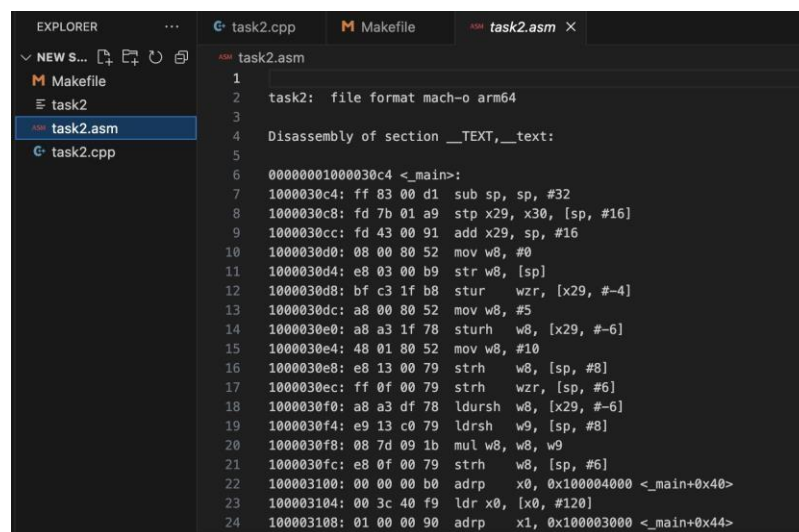
Create a 'Makefile' similar to the one used in the first task, with targets for 'all', 'clean', 'dump', and 'run'.

Makefile yang saya buat:



```
1 all: task2
2
3 task2: task2.cpp
4     g++ -o task2 task2.cpp
5
6 dump: task2
7     objdump -d task2 > task2.asm
8
9 clean:
10    rm -f task2 task2.asm
11
12 run: task2
13    ./task2
14
```

Menjalankan command `make dump`. Ini akan disassemble file executable `task2` dan menyimpan hasilnya ke dalam file `task2.asm`. Dan terlihat sudah ada file `task2.asm` dan assembly code dari program c++ yang dibuat.



```
1 task2: file format mach-o arm64
2
3
4 Disassembly of section __TEXT,__text:
5
6 00000001000030c4 <_main>:
7 1000030c4: ff 83 00 d1 sub sp, sp, #32
8 1000030c8: fd 7b 01 a9 stp x29, x30, [sp, #16]
9 1000030cc: fd 43 00 91 add x29, sp, #16
10 1000030d0: 08 00 80 52 mov w0, #0
11 1000030d4: e8 03 00 b9 str w0, [sp]
12 1000030d8: bf c3 1f b8 stur wzr, [x29, #-4]
13 1000030dc: a8 00 80 52 mov w0, #5
14 1000030e0: a8 a3 1f 78 sturh w0, [x29, #-6]
15 1000030e4: 48 01 80 52 mov w0, #10
16 1000030e8: e8 13 00 79 strh w0, [sp, #8]
17 1000030ec: ff 0f 00 79 strh wzr, [sp, #6]
18 1000030f0: a8 a3 df 78 ldursh w0, [x29, #-6]
19 1000030f4: e9 13 c0 79 ldursh w0, [sp, #8]
20 1000030f8: 08 7d 09 1b mul w0, w0, w9
21 1000030fc: e8 0f 00 79 strh w0, [sp, #6]
22 100003100: 00 00 00 b0 adrp x0, 0x100004000 <_main+0x40>
23 100003104: 00 3c 40 f9 ldr x0, [x0, #120]
24 100003108: 01 00 00 90 adrp x1, 0x100003000 <_main+0x44>
```