Percobaan I
Compiler Bahasa C dan Bahasa Assembly Intel® x86

Ahmad Aziz (13220034)
Asisten: Oktavian Putra M (13219039)
Tanggal Percobaan : 23/09/2022
EL3111 Praktikum Arsitektur Sistem Komputer
Laboratorium Sinyal dan Sistem – Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

Abstrak*—Menjalankan program dalam bahasa pemrograman C memerlukan alat untuk melakukan kompilasi program yang disebut sebagai compiler. Pada praktikum ini akan dilakukan percobaan compiler bahasa C dan juga bahasa assembly. Melakukan setiap tahapan kompilasi mulai dari preprocessing, compiling, assembling, dan juga linking. Dilakukan juga berbagai optimisasi pada program yang dikompilasi. Pada praktikum ini juga dilakukan proses disassembly, disassembly merupakan proses untuk mendiassable binary file menjadi file assembly. Pada praktikum ini juga dilakukan percobaan untuk menggunakan kompilasi perintah menggunakan batch file. Kompilasi perintah ini berisi kumpulan perintah yang akan dieksekusi ketika file tersebut dijalankan. Kompilasi perintah ini bertujuan untuk mempermudah dan mempercepat dalam melakukan perintah yang banyak dan berulalang atau untuk file yang lebih dari satu. Kompilasi perintah ini dapat dilakukan dengan batch file atau dengan makerfile. Terakhir, pada percobaan modul ini akan mempelajari tentang bahasa assembly hasil kompilasi dan juga mekanisme penyimimpanan memory stack pada komputer. Struktur memory dapat dianalisis dan direkonstruksi dengan melihat hex file dengan software HexEdit yang akan digunakan pada praktikum ini. Sedangkan untuk menganalisis assembly dan kompilasi akan dilakukan dengan debugger yang akan menggunakan codeblocks.

***Kata Kunci—Compiler*, Intel, Assembly, *stack*, memori.**

I. PENDAHULUAN

Pada praktikum modul 1 yaitu Compiler Bahasa C dan Bahasa Assembly Intel® x86 dilakukan sebanyak sembilan percobaan yang bertujuan diantaranya sebagai berikut:

- Praktikan memahami tahap-tahap kompilasi program dalam bahasa C sebagai bahasa tingkat tinggi hingga diperoleh bahasa tingkat rendah yang dapat dieksekusi oleh mesin.
- Praktikan mampu melakukan kompilasi program bahasa C menggunakan compiler GCC beserta penggunaan makefile dan batch file.
- Praktikan memahami bahasa assembly dan mampu melakukan analisis terhadap bahasa assembly Intel® x86 yang dihasilkan oleh compiler GCC.
- Praktikan memahami penggunaan stack memory pada setiap procedure call.

Praktikum pada modul ini menggunakan bahasa pemrograman C dengan compiler GCC untuk melakukan percobaan dan juga bahasa assembly sebagai hasil compiling dari bahasa C yang akan dianalisis. Ada beberapa topik yang dibahas pada modul praktikum ini yaitu sebagai berikut:

a. Kompilasi dengan GCC.
b. *Disassembly* dengan GCC.
c. Optimisasi program pada proses kompilasi.
d. Makefile dan Batch file.
e. Instruksi dan Bahasa Assembly Intel® x86.
f. *Stack* dan *procedure call*.

Dalam melakukan percobaan dan analisis pada praktikum modul ini, perangkat lunak dan alat yang digunakan adalah sebagai berikut:

a. Compiler GCC
b. CodeBlocks
c. Code editor Visual Studio Code
d. HexEdit

II. LANDASAN TEORETIS

Bahasa pemrograman C merupakan bahasa pemrograman yang bersifat processor independent. Artinya bahasa pemrograman C dapat berjalan tanpa bergantung pada jenis ataupun arsitektur processor dimana program tersebut dijalankan. Karena itu, bahasa C dapat dijalankan pada berbagai mesin tanpa mengubah kodenya. Karena flexibilitas dan portabilitasnya inilah bahasa pemrograman C banyak dipakai dalam pemrograman sistem ebeded. Bahasa pemrograman C dapat berjalan pada berbagai mesin karena compiler yang mendukung pada setiap mesin yang menjalankannya. Mesin tidak dapat membaca bahasa C, mesin hanya dapat membaca bahasa mesin (binary). Oleh karena itu, dibutuhkan sebuah proses agar bahasa C dapat dibaca oleh mesin, proses ini disebut proses kompilasi.

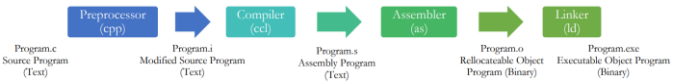
A. Kompilasi Menggunakan GCC

Untuk membuat suatu program, bahasa tingkat tinggi

cenderung banyak digunakan karena bahasa tingkat tinggi ini mudah dimengerti oleh manusia seperti halnya bahasa C. Sayangnya, bahasa tingkat tinggi tidak dapat dimengerti oleh mesin (mikroprosesor) sehingga tidak dapat dieksekusi. Oleh karena itu, diperlukan sebuah penerjemah bahasa pemrograman tingkat tinggi menjadi bahasa tingkat rendah yang berisi urutan instruksi yang dimengerti oleh mesin. Urutan instruksi tersebut kemudian dikemas dalam suatu bentuk executable object program yang disimpan dalam bentuk file biner.

Kumpulan instruksi yang dimengerti oleh mesin disebut instruction set. Dari sisi instruction set, terdapat dua penggolongan mesin (mikroprosesor) yaitu complex instruction set computer (CISC), contohnya mikroprosesor Intel®, dan reduced instruction set computer (RISC), contohnya MIPS32®. Beberapa mikroprosesor Intel® memiliki tambahan set instruksi seperti MMX dan SSE.

Proses menerjemahkan baris kode program dalam bahasa C menjadi file executable dilakukan dalam empat langkah yaitu preprocessor, compiler, assembler, dan linker yang seluruhnya disebut sistem kompilasi.



a. Preprocessor

Semua perintah preprocessor yang ditulis dalam bahasa tingkat tinggi akan diproses terlebih dahulu oleh preprocessor sebelum compiler melaksanakan tugasnya. Beberapa tugas dari preprocessor ini adalah sebagai berikut.

- Semua komentar dalam file program diganti dengan spasi satu buah.
- Semua \n (backslash-newline) yang menandakan baris baru akan dihapus tidak peduli dimanapun dia berada. Fitur ini memungkinkan kita untuk membagi baris program yang panjang ke dalam beberapa baris tanpa mengubah arti.
- Macro yang telah didefinisikan diganti dengan definisinya.

Contohnya, pada perintah #define MAX_ROWS 10, preprocessor akan mengganti semua kata MAX_ROWS dengan 10. Pada perintah #include , preprocessor akan mengganti baris tersebut dengan isi file stdio.h.

Untuk melakukan preprocessor dengan GCC adalah dengan menjalankan perintah berikut pada terminal:

```
gcc -E Program.c
```

Eksekusi perintah tersebut akan menampilkan di layar Command Prompt kode Program.c setelah melalui proses preprocessing. Agar memperoleh output berupa file, dapat menggunakan tambahan perintah sebagai berikut.

```
gcc -E Program.c > Program.i
```

Eksekusi perintah tersebut akan menghasilkan file Program.i berisi kode Program.c yang telah melalui preprocessing pada folder yang sama dengan file Program.c. File ini dapat dibuka dengan teks editor.

b. Compiler

Compiler akan menerjemahkan bahasa tingkat tinggi C menjadi kode assembly. Kode assembly ini berisi instruksi-instruksi yang sesuai dengan instruction set yang dimiliki oleh mesin. File yang dihasilkan pada tahap ini masih berupa file teks (.s).

Untuk melakukan proses compiling, perlu dilakukan proses processing terlebih dahulu, untuk melakukan proses preprocessing dan compiling menggunakan GCC dapat menggunakan perintah berikut pada terinal:

```
gcc -S Program.c
```

c. Assembler

Assembler akan menerjemahkan bahasa assembly menjadi file objek. File objek ini merupakan file biner (.o).

Untuk melakukan proses *assembly* dapat menggunakan perintah berikut dengan GCC pada terminal yang mana akan melakukan proses sebelum assembly juga:

```
gcc -c Program.c
```

Eksekusi perintah tersebut akan menghasilkan file Program.o yang merupakan file biner. File ini dapat dibuka dengan program hex editor contohnya HexEdit.

d. Linker

Linker akan menggabungkan file biner yang diperoleh pada tahap sebelumnya dengan file biner lain yang merupakan dependency dari program yang dibuat, contohnya library untuk menjalankan fungsi printf. Hasil dari linker berupa file biner executable (dalam platform Microsoft® Windows™, file ini memiliki akhiran .exe).

Untuk melakukan proses linker dapat menggunakan perintah berikut yang akam melakukan proses sebelum linking juga (*preprocessing, compiling, assembly*):

```
gcc -o Program.exe Program.c
```

Eksekusi perintah tersebut akan menghasilkan Program.exe yang dapat langsung dieksekusi (dijalankan). Kita juga dapat melakukan kompilasi dua file bahasa C sekaligus dengan perintah berikut:

```
gcc -o Program.exe sub.c main.c
```

B. Disassembly Menggunakan GCC

Selain dapat melakukan kompilasi, paket compiler GCC juga menyertakan sebuah disassembler yang mampu melakukan disassembly file biner (.o atau .exe) menjadi file assembly (.s) bernama Object Dump. Untuk melakukan disassembly, kita dapat menggunakan perintah berikut.

```
objdump -d Program.o
objdump -d Program.exe
```

Hasil dari proses disassembly ditampilkan pada jendela Command Prompt. Agar hasil dari proses disassembly dapat disimpan ke dalam suatu file, kita dapat menggunakan perintah berikut.

```
objdump -d Program.o > Program.s
objdump -d Program.exe > Program.s
```

Dengan demikian, hasil proses disassembly akan disimpan dalam file Program.s.

C. Optimisasi Program melalui Proses Kompilasi

GCC mendukung beberapa tingkat optimisasi program yang dapat dilakukan saat proses kompilasi dilakukan. Terdapat beberapa tingkat optimisasi program yang dapat dipilih dengan menambahkan flag optimisasi saat melakukan kompilasi

program. Umumnya optimisasi program merupakan trade-off antara execution speed, program size, compilation time, dan kemudahan dalam melakukan debugging.

```
gcc –O2 -o Program.exe Program.c
```

Flag –O2 tersebut menandakan bahwa proses kompilasi dilakukan dengan optimisasi tingkat dua.

Beberapa flag optimisasi yang dikenali oleh GCC adalah –O0, –O1, –O2, –O3, –Os, dan –Ofast. Perbedaan masing-masing level optimisasi diberikan sebagai berikut.

Opsi	Level Optimisasi	Waktu Eksekusi	Ukuran Kode	Pemakaian Memory	Waktu Kompilasi
-O0	Optimisasi untuk mempercepat waktu kompilasi (*default*)	+	+	-	-
-O1 -O	Optimisasi untuk ukuran kode dan waktu eksekusi	-	-	+	+
-O2	Optimisasi lebih tinggi untuk ukuran kode dan waktu ekseksui	--	O	+	++
-O3	Optimisasi lebih tinggi lagi untuk ukuran kode dan waktu eksekusi	---	O	+	+++
-Os	Optimisasi untuk ukuran kode	O	--	+	++
-Ofast	Sama dengan -O3 namun ditambah optimisasi terhadap fungsi-fungsi matematika yang tidak perlu akurasi tinggi	---	O	+	+++

Note: + lebih tinggi; ++ lebih lebih tinggi; +++ lebih lebih lebih tinggi; O tidak berubah; - lebih sedikit; -- lebih lebih sedikit; --- lebih lebih lebih sedikit.

D. Makefile dan Batch File

Untuk suatu project yang terdiri atas beberapa file kode, tentu akan sangat merepotkan untuk melakukan kompilasi dengan menggunakan perintah kompilasi yang ditulis pada command prompt satu per satu untuk setiap file. GCC memiliki fitur makefile yang berfungsi untuk menulis daftar nama file kode di dalam project tersebut. Kita cukup memberikan GCC nama makefile lalu GCC akan melakukan proses kompilasi untuk semua file tersebut untuk kemudian menggabungkannya pada file executable. Makefile dapat bersifat sederhana hingga kompleks, bergantung pada sejauh mana kita menggunakan makefile untuk mengorganisasikan project kita. Contoh isi dari makefile adalah sebagai berikut.

```
all: contoh
contoh: main.o text.o
    gcc main.o text.o -o contoh.exe
main.o: main.c
    gcc -c main.c
text.o: text.c
    gcc -c text.c
```

GCC dapat diperintahkan untuk melakukan kompilasi makefile dengan perintah sebagai berikut.

```
mingw32-make -f makefile
```

Perintah tersebut akan melakukan kompilasi terhadap makefile yang diberikan menjadi sebuah program bernama contoh.exe. Program ini dihasilkan oleh hasil linker terhadap dua file objek bernama main.o dan text.o (tentunya termasuk dengan library yang lain yang dibutuhkan). Untuk memperoleh main.o, GCC harus melakukan kompilasi source code main.c menjadi file objek. Begitupula untuk memperoleh text.o, GCC harus melakukan kompilasi source code text.c.

Pada platform Microsoft® Windows™, terdapat sebuah file shell script bernama Windows™ Batch File. Kita dapat menuliskan perintah-perintah yang biasa kita tuliskan secara terpisah pada command prompt dalam suatu file yang disimpan dengan ekstensi .bat. Untuk mengeksekusi perintah-perintah tersebut, kita cukup menjalankan file .bat tersebut sehingga command prompt terbuka dan perintahperintah yang kita tuliskan dieksekusi secara otomatis. Contoh Windows™ Batch File adalah sebagai berikut.

```
%~d0
cd "%~dp0"
gcc -O2 -E code.c > Program.l
gcc -O2 -S code.c
gcc -O2 -c code.c
gcc -O2 -o code.c
pause
objdump -d code.o > dump_o.dmp
objdump -d prog.exe > dump_exe.dmp
pause
```

Windows™ Batch File tersebut berisi perintah sebagai berikut. Perintah %~d0 memerintahkan command prompt untuk berpindah drive letter ke drive letter yang sesuai dengan lokasi Windows™ Batch File berada. Selanjutnya, perintah cd "%~dp0" memerintahkan command prompt untuk berpindah folder ke lokasi Windows™ Batch File berada. Selanjutnya, command prompt mengeksekusi perintah yang memanggil GCC secara berurutan hingga berhenti akibat adanya perintah pause. Untuk melanjutkan eksekusi, kita cukup menekan sebarang tombol pada keyboard.

E. Instruksi dan Bahasa Assembly Intel® x86

Arsitektur mikroprosesor Intel® x86 merupakan salah satu arsitektur mikroprosesor yang banyak digunakan. Dengan mempelajari bahasa assembly dan instruksi Intel® x86, kita akan sangat terbantu dalam melakukan proses debugging dan optimisasi program yang kita buat. Dalam mikroprosesor Intel® x86, terdapat banyak register yang dapat digunakan. Namun, pada praktikum kali ini, kita cukup mempelajari beberapa register berikut.

- EAX, EBX, ECX, dan EDX adalah register 32-bit yang bersifat general storage.
- ESI dan EDI adalah register 32-bit yang digunakan sebagai indexing register. Register ini juga dapat digunakan sebagai general storage.
- ESP adalah register 32-bit yang digunakan sebagai stack pointer. Dengan demikian, ESP akan berisi nilai alamat (address) elemen puncak (top element) dari stack. Perlu diingat bahwa stack membesar dari alamat tinggi (high address) ke arah alamat rendah (low address). Dengan demikian, memasukkan elemen baru ke dalam stack akan mengurangi nilai alamat yang tersimpan pada ESP sedangkan mengeluarkan elemen dari dalam stack akan menambah ESP.
- EBP adalah register 32-bit yang digunakan sebagai base pointer. Dengan demikian, EBP akan berisi alamat dari current activation frame pada stack.
- EIP adalah register 32-bit yang digunakan sebagai instruction pointer. Dengan demikian, EIP akan berisi alamat dari instruksi selanjutnya yang akan dieksekusi.

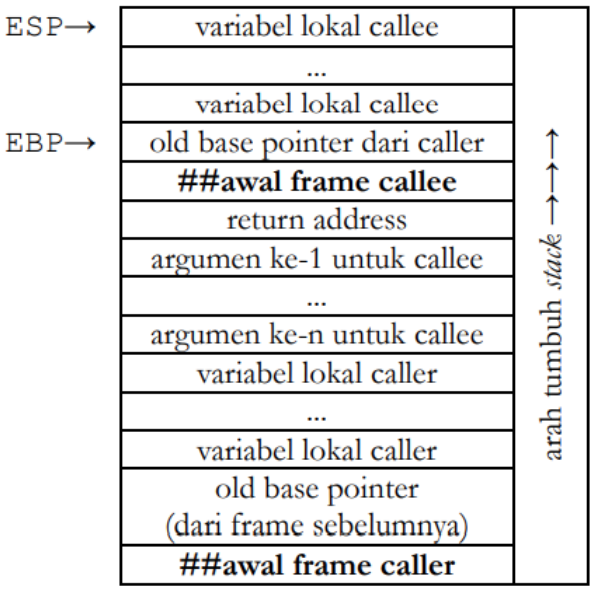
Instruksi-instruksi yang digunakan pada Intel® x86 tidak akan dijelaskan di dalam modul praktikum ini. Praktikan dapat mempelajari lebih jauh mengenai instruksi-instruksi ini pada

bab 3 di buku "Computer System – A Programmer's Perspective" yang ditulis oleh Bryant dan O'Hallaron.

F. Stack dan Procedure Call

Stack pada umumnya disusun atas beberapa activation frame. Setiap frame memiliki sebuah base pointer yang menunjukkan alamat tertinggi (highest address) pada frame tersebut. Karena stack tumbuh dari high address menuju low address, base pointer akan menunjukkan alamat tertinggi frame tersebut.

Ketika suatu program (caller) memanggil sebuah prosedur (callee), caller akan memasukkan argumen-argumen untuk memanggil callee dari argumen terakhir hingga argumen paling awal secara berurutan ke dalam stack. Selanjutnya, caller akan memasukkan return address ke dalam stack. Kemudian, callee memasukkan alamat old base pointer milik caller ke dalam stack dan memperbarui nilai base pointer yang sesuai dengan frame callee (nilai base pointer yang baru sama dengan nilai stack pointer setelah old base pointer disimpan ke dalam stack). Kemudian callee melakukan alokasi terhadap variabel lokal dan melakukan komputasi sesuai dengan fungsi callee tersebut.



Ketika callee selesai dieksekusi, callee akan menyimpan return value pada register EAX. Kemudian, callee akan membersihkan framenya sendiri dengan mengganti alamat base pointer dengan old base pointer yang telah disimpan pada stack. Kemudian, return address digunakan untuk melanjutkan eksekusi instruksi pada caller.

III. HASIL DAN ANALISIS

Setelah melakukan percobaan pada semua tugas didapatkan hasil sebagai brerikut:

A. Tugas I : Proses Kompilasi Bahasa C Menggunakan GCC

Pada tugas 1 dilakukan proses kompilasi pada program Bahasa C. Kode program bahasa C yang digunakan dan hasil kompilasi pada percobaan ini dapat dilihat pada lapiran tugas 1. Proses kompilasi yang dilakukan secara berurutan mulai dari proses preprocessor, compiler, assembler, dan linker. Semua proses kompilasi pada percobaan 1 ini dilakukan dengan perintah command line satu per satu pada terminal powershell windows dengan GCC.

Proses kompilasi pertama yang dilakukan adalah proses preprosessing. Proses preprocessing yang bertujuan untuk mempersiapkan kode dalam bahasa C untuk dilakukan proses compiling. Pada hasil preprocessing pada kode program code.c terlampir, maka didapatkan hasil dalam bentuk file .i sebagai berikut:

```
# 0 "code.c"
# 0 "<built-in>"
# 0 "<command-line>"
# 1 "code.c"
# 13 "code.c"
int main(void){
    int indeks;
    int accumulator;
    indeks = 0;
    accumulator = 0;
    while(indeks<500){
        accumulator = accumulator + indeks;
        indeks = indeks + 1;
    }
    return accumulator;
}
```

Dapat dilihat pada kode hasil preprocessing diatas masih seperti bahasa program C yang dapat dipahami seperti biasa. Namun, terdapat perbedaan pada bagian header baris kode dimana tidak terdapat lagi komentar pada headernya. Hal ini menunjukkan pada proses preprocessing ini terjadi penyederhanaan baris kode program C dimana semua komentar pada baris program dihilangkan sehingga terjadi pengurangan jumlah baris yang sebelumnya berjumlah 23 baris termasuk komentar menjadi hanya 16 baris. Dapat diperhatikan juga pada bagian header program terdapat baris baru yang sebelumnya tidak ada pada program bahasa C yang ditulis.

Setelah dilakukan preprocessing, selanjutnya proses dilanjutkan dengan proses compiling. Proses compiling ini akan memproses output dari proses preprocesesing yang telah dilakukan, Proses compiling menghasilkan kode assembly. Pada percobaan ini masih menggunakan kode program yang sama yaiu code.c , setelah dilakukan compiling menghasilkan kode assembly dalam format file .s seperti yang terdapat pada lampiran code lapiran tugas 1.

Proses selanjutnya adalah assembly. Pada proses assembly, kode akan diterjemahkan ke bahasa assembly. Kode hasil proses assembly dapat di export kedalam bentuk file .s.

Preses terakhir dalam rangkaian proses compiling adalah proses linking. Proses lingking dilakukan untuk mengubah bahasa assembly menjadi bahasa mesin (binary code) yang dapat diexport dalam format .o atau .exe yang dapat dijalankan/dieksekusi oleh mesin.

B. Tugas 2 : Proses Kompilasi Bahasa C Menggunakan GCC dengan Bantuan Batch File

Pada percobaan tugas 2 ini akan dilakukan proses compilasi dengan GCC menggunakan batch file. Kode yang digunakan masih sama dengan tugas yaitu code.c yang terdapat pada

lampiran. Berikut adalah baris kode dalam batch file yang digunakan:

```
%~d0
cd "%~dp0"
gcc -E code.c > code.i
gcc -S code.c
gcc -c code.c
gcc -o code.exe code.c
code.exe
pause
```

Pada batch file tersebut dilakukan berbagai proses complilasi dari preprocessing hingga linking. Batchfile akan mengeksekusi baris-baris perintah dalam batchfile yang sudah kita tulis untuk melakukan proses kompilasi. Hasil dari eksekusi file batch ini manghasilkan file output yang identik/sama persis dengan output hasil kompilasi melalui terinal command line yang ada pada tugas satu.

Dengan menggunakan batch file proses kompilasi yang dilakukan lebih mudah karena dapat dengan sekali eksekusi untuk menjalankan semua perintah kompilasi yang diinginkan.

File output hasil eksekusi dari batchfile ini terdapat pada folder tugas 2.

C. Tugas 3 : Disassembly File Objek

GCC selain dapat melakukan assembly program juga dapat melakukan proses disassembly. Proses disassembly adalah proses untuk mengembalikan file program dalam bentuk binary code (.o atau .exe) menjadi kembali ke bentuk kode dalam bahasa assembly.

Pada percobaan ini dilakukan percobaan untuk mendisassembly kode program dalam bentuk binary code yang sudah decompile pada percobaan sebelumnya.

D. Tugas 4: Optimisasi Kompilasi Program pada GCC

Proses compilasi yang dilakukan menggunakan GCC dilakukan secara otomatis menerjemahkan kode dari bahasa program C ke bahasa mesin. Proses kompilasi ini dapat kita kustomisasi untuk menghasilkan kode bahasa mesin untuk mendapatkan hasil yang terbaik sesuai dengan kebutuhan/keinginan kita. Proses ini disebut dengan optimisasi program.

Pada perobaan ini dilakukan berbagai jenis optimisasi pada proses kompilasi program bahasa C. Code program bahasa C yang digunakan pada proses optimisasi ini masih sama dengan percobaan sebelumnya. Dilakukan parameterisasi untuk setiap kode program yang dioptimisasi dengan cara tertentu dengan menabahkan jenis optimisasi pada akhir nama program sepert pada folder lab yang ada pada program.

Berdasarkan hasil optimisasi pada semua program untuk setiap jenis optimisasi tersebut didapatkan output yang ketika dijalankan secara fungsionalitasnya identik karena merupakan kode program yang sama.

Berikut adalah perbandingan ukuran file untuk semua output binary pada semua jenis optimisasi:

- O0 : 768 byte
- O1 : 732 byte
- O2 : 896 byte
- O3 : 896 byte
- Os : 896 byte
- Ofast : 896 byte

Dapat diperhatikan dari data hasil optimisasi tersebut file hasil compilasi terkecil adalah dengan optimisasi O1 dilanjutkan O0. Sedangkan jenis optimisasi lainnya menghasilkan ukuran file yang sama.

Hal ini berbanding lurus dengan jumlah baris kode pada bahasa assembly yang dihasilkan. Untuk optimisasi O2 sampai Ofast jumlah baris kode assemblynya sama yaitu 25 baris kode, dan yang terkecil yaitu O1 dengan jumlah baris kode paling sedikit yaitu hanya 18 baris. Sedangkan, pengecualian untuk optimisasi O0 yang tidak berbanding lurus, dimana besaran filenya lebih kecil namun jumlah baris assemblynya lebih besar dari keempat optmisasi yang sama lainnya yaitu sebanyak 30 baris.

E. Tugas 5: Kompilasi Beberapa File Kode dengan GCC

Kompilasi beberapa file yaitu mengkompilasi file yang saling terhubung pada rangkaian program C yang biasa disebut dengan file header. File header ini dapat menggabungkan beberapa file program C. Ini biasanya digunakan pada program yang kompleks untuk mempermudah dalam debugging dan pembacaan kode. Penggabungan file ini juga banyak ditemukan dalam library kode dalam program terintegrasi.

Pada percobaan ini dilakukan kompilasi beberapa program yang dihubungkan dengan file header .h. program yang digunakan sebagai main program adalah sebagai berikut:

```
// Praktikum EL3111 Arsitektur Sistem
Komputer
// Modul : 1
// Percobaan : 5
// Tanggal : 23 September 2022
// Kelompok : 10
// Rombongan : B
// Nama (NIM) 1 : Ahmad Aziz (13220034)
// Nama (NIM) 2 : Gilbert Ng (13220032)
// Nama File : main_text.c
// Deskripsi : Demonstrasi MakeFile
// Memanggil prosedur test pada text.c
#include "text.h"
void main(void){
    test();
}
```

Program main_text.c ini akan memanggil file headernya dengan #include. Dan pada fungsi main akan memanggil fungsi yang ada dalam fil text.c yang dhibungkan oleh file header tersebut.

Berikut adalah kode yang ada pada file header:

```
// Praktikum EL3111 Arsitektur Sistem
Komputer
// Modul : 1
// Percobaan : 5
// Tanggal : 23 September 2022
```

```
// Kelompok : 10
// Rombongan : B
// Nama (NIM) 1 : Ahmad Aziz (13220034)
// Nama (NIM) 2 : Gilbert Ng (13220032)
// Nama File : text.h
// Deskripsi : Demonstrasi MakeFile,
Mencetak string ke layar
#ifndef TES_H
    #define TES_H 100
    void test(void);
#endif
```

Sedangkan untuk file c nya sendiri adalah sebagai berikut:

```
// Praktikum EL3111 Arsitektur Sistem
Komputer
// Modul : 1
// Percobaan : 5
// Tanggal : 23 September 2022
// Kelompok : 10
// Rombongan : B
// Nama (NIM) 1 : Ahmad Aziz (13220034)
// Nama (NIM) 2 : Gilbert Ng (13220032)
// Nama File : text.c
// Deskripsi : Demonstrasi MakeFile,
Mencetak string ke layar
#include <stdio.h>
#include "text.h"
void test(void){
    printf("Arsitektur Sistem Komputer
sangat menyenangkan!\n");
}
```

GCC dapat melakukan kompilasi pada semua file tersebut dan menjadikannya satu kesatuan yang nantinya dalam bentuk binary code yang dapat di eksekusi.

Perintah kompilasi untuk melakukan compilasi bersama adalah sebagai berikut:

```
gcc -o main_text.exe text.c main_text.c
```

Perintah tersebut akan mengkompilasi file dan akan menghasilkan satu binary code yaitu main_text.exe. Setelah selesai kompilasi dan file binary dieksekusi program yang didapatkan berjalan dengan baik.

F. Tugas 6: Penggunaan Makefile pada GCC

Kompilasi dapat dilakukan beberapa perintah secara bersamaan. Hal ini juga dapat dilakukan dengan menggunakan batch file seperti yang dilakukan pada percobaan 2. Pada percobaan ke 5 ini akan dilakukan proses kompilasi beberapa perintah kompilasi dengan menggunakan makefile pada GCC.

Berikut ini adalah perintah kompilasi yang digunakan pada makefile:

```
all: main_text.exe
main_text.exe: main_text.o text.o
    gcc main_text.o text.o -o main_text.exe
main_text.o: main_text.c
    gcc -c main_text.c
text.o: text.c
    gcc -c text.c
```

Program yang digunakan pada percobaan ini adalah program yang sama dengan program percobaan sebelumnya. Pada percobaan ini setelah dilakukan kompilasi dengan makefile, hasil output binary filenya memiliki ukuran yang sama dan ketika dijalankan juga menghasilkan output dan fungsionalitas yang sama sesuai dengan program. Artinya kedua cara kompilasi ini menghasilkan output yang identik.

Yang membedakan kedua cara ini adalah dengan makefile perintah kompilasi dijalankan satu kali saja ketika kita mengeksekusi makefile tersebut yang hampir mirip dengan batchfile pada percobaan sebelumnya. Sedangkan, dengan menggunakan perintah terminal kita mengetikkan baris perintah dan mengeksekusinya melalaui terminal. Ini menjadi kelebihan pernggunaan makefile untuk meminimalisir terjadinya kesalah serta mempermudah dalam melakukan kompilasi karena kita dapat menyimpan dan mengedit kembali makefile yang sudah dibuat dan mengonfigurasi ulangnya jika dibutuhkan tanpa menulis ulang.

Perbedaan antara makefile dan juga batch file ada pada cara kerjanya. Pada prinsinya batchfile seperti terminal windows yang dibuat dalam bentuk file, dia akan mengeksekusi baris per baris perintah yang ada di dalamnya. Jadi bentuknya akan sama saja dengan terminal windows. Sedangkan makefile merupakan tools eksternal yang dapat menjalankan perintah yang sesuai dengan cara penulisan makefile itu sendiri.

Hasil kompilasi semua cara ini sama karena proses kompilas ini dilakukan pada GCC, makefile, terminal, dan juga batcfile hanya menjalankan perintah GCC untuk menjalankan kompilasi.

G. Tugas 7: Header File

Header file merupakan file yang digunakan untuk menghubungkan file program C lain yang akan digunakan dan digabungkan pada suatu program yang sama. Percobaan headerfile ini menggunakan 2 program C yaitu program main dan program fungsi add yang dihubungkan oleh file header .h.

Berikut ini adalah kode fungsi add yang digunakan pada percobaan:

```
// Praktikum EL3111 Arsitektur Sistem
Komputer
// Modul : 1
// Percobaan : 7
// Tanggal : 23 September 2022
// Kelompok : 10
// Rombongan : B
// Nama (NIM) 1 : Ahmad Aziz (13220034)
// Nama (NIM) 2 : Gilbert Ng (13220032)
// Nama File : add.c
// Deskripsi : Demonstrasi header file
// Menjumlahkan dua bilangan
```

```
#define START_VAL 0

int accum = START_VAL;
int sum(int x, int y){
    int t = x + y;
    accum += t;
    return t;
}
```

Dan berikut ini adalah kode main yang digunakan yang akan memanggil fungsi sum pada kode yang ada di file add.c

```
// Praktikum EL3111 Arsitektur Sistem
Komputer
// Modul : 1
// Percobaan : 7
// Tanggal : 23 September 2022
// Kelompok : 10
// Rombongan : B
// Nama (NIM) 1 : Ahmad Aziz (13220034)
// Nama (NIM) 2 : Gilbert Ng (13220032)
// Nama File : main.c
// Deskripsi : Demonstrasi header file
// Main program
#include <stdio.h>
#include "add.h"
int main(){
    int a;
    int b;
    scanf("%d %d", &a, &b);
    int c = sum(a,b);
    printf("Hasil penjumlahan %d + %d =
%d",a,b,c);
    return 0;
}
```

Kedua file ini dihubungkan oleh file header atau main akan menginclude file add.h melalui headernya untuk memanggil file .c nya. Berikut adalah file header dari add

```
// Praktikum EL3111 Arsitektur Sistem
Komputer
// Modul : 1
// Percobaan : 7
// Tanggal : 23 September 2022
// Kelompok : 10
// Rombongan : B
// Nama (NIM) 1 : Ahmad Aziz (13220034)
// Nama (NIM) 2 : Gilbert Ng (13220032)
// Nama File : add.h
// Deskripsi : Demonstrasi header file
// Header file fungsi add
#ifndef ADD_H
    int sum(int x, int y);
#endif
```

Percobaan ini hampir sama dengan percobaan sebelumya, pada percobaan ini akan dikompilasi dengan perintah berikut melalui terminal:

```
gcc -o main.exe add.c main.c
```

Setelah dilakukan eksekusi program pada file output binary didapatkan hasil yang berjalan sesuai dengan program C yang telah dibuat.

H. Tugas 8: Pemanggilan Prosedur dan Stack Memory

Pemanggilan prosedur atau fungsi berkaitan dengan register dan memori yang ada pada computer.

I. Tugas 9: Program Fibonacci

Berikut ini adalah kode main dari program fibbonaci

```
// Praktikum EL3111 Arsitektur Sistem
Komputer
// Modul : 1
// Percobaan : 9
// Tanggal : 23 September 2022
// Kelompok : 10
// Rombongan : B
// Nama (NIM) 1 : Ahmad Aziz (13220034)
// Nama (NIM) 2 : Gilbert Ng (13220032)
// Nama File : fibo_main.c
// Deskripsi : Main file untuk fibonacci
function
// Menghitung deret fibonacci sebanyak
input user

#include "inputn.h"
#include "fibo.h"
#include <stdio.h>
int main(){
    int i;
    int N = inputn();
    for(i = 0; i < N; i++){
        printf("%d ", fibonacci(i));
    }
    return 0;
}
```

Pada program ini memanggil dua file lainnya yang berisi fungsi input dan fibbonacinya sendiri dengan masing masing-masing headerfile sebagai berikut:

Fibo.h

```
// Praktikum EL3111 Arsitektur Sistem
Komputer
// Modul : 1
// Percobaan : 9
```

```
// Tanggal : 23 September 2022
// Kelompok : 10
// Rombongan : B
// Nama (NIM) 1 : Ahmad Aziz (13220034)
// Nama (NIM) 2 : Gilbert Ng (13220032)
// Nama File : inputn.h
// Deskripsi : Deklarasi fungsi fibonacci
// Header file library fibonacci

int fibonacci(int n);
```

fibo.c

```
// Praktikum EL3111 Arsitektur Sistem
Komputer
// Modul : 1
// Percobaan : 9
// Tanggal : 23 September 2022
// Kelompok : 10
// Rombongan : B
// Nama (NIM) 1 : Ahmad Aziz (13220034)
// Nama (NIM) 2 : Gilbert Ng (13220032)
// Nama File : fibo.h
// Deskripsi : Library fibonacci function
// Menghitung deret fibbonaci

#include "fibo.h"

int fibonacci(int n){
    if(n <= 1){
        return 1;
    }
    else{
        return fibonacci(n-1) +
fibonacci(n-2);
    }
}
```

Kemudian fungsi untuk input yang juga memvalidasi input:

Inputn.h

```
// Praktikum EL3111 Arsitektur Sistem
Komputer
// Modul : 1
// Percobaan : 9
// Tanggal : 23 September 2022
// Kelompok : 10
// Rombongan : B
// Nama (NIM) 1 : Ahmad Aziz (13220034)
// Nama (NIM) 2 : Gilbert Ng (13220032)
// Nama File : inputn.h
// Deskripsi : Deklarasi fungsi inputn
// Header file library inputn

int inputn();
```

dan file c nya:

```
// Praktikum EL3111 Arsitektur Sistem
Komputer
// Modul : 1
// Percobaan : 9
// Tanggal : 23 September 2022
// Kelompok : 10
// Rombongan : B
// Nama (NIM) 1 : Ahmad Aziz (13220034)
// Nama (NIM) 2 : Gilbert Ng (13220032)
// Nama File : inputn.c
// Deskripsi : Fungsi input dan validasi
input user
// Fungsi input dan validasi input user

#include "inputn.h"
#include <stdio.h>

int inputn(){
    int n;
    printf("masukkan n: ");
    scanf("%d", &n);
    while (n<2){
        printf("masukkan n: ");
        scanf("%d", &n);
    }
    return n;
}
```

IV. Simpulan

- Bahasa C merupakan bahasa yang processor atau OS independent yang dapat berjalan pada platform yang berbeda karena adanya proses ompiling untuk menerjemahkan kode ke platformnya.
- Bahasa program C dapat dikompile dan dikostomisasi jenis kompilasinya dengan optimisasi dan akan mendapatkan hasil yang sesuai dengan kebutuhan dengan jenis optimisasi yang sesuai.
- Kompilasi dapat dilakukan bersamaan untuk banyak perintah dengan menggunakan batfile atau makefile.
- Bahasa C dapat menggabungkan beberapa file yang terpisah dengan tujuan untuk kerapihan kode dan kemudahan pembacaan kode dengan header file.

Referensi

Basic format for books:

[1] Bryant, Randal, dan David O'Hallaron. *Computer Systems: A Programmer's Perspective 2nd Edition*. 2011. Massachusetts: Pearson Education Inc.

[2] Patterson, David, dan John Hennessy. *Computer Organization and Design: The Hardware/Software Interface*. 2012. Waltham: Elsevier Inc.

Lampiran

1) *Source code untuk tugas 1 dan 2*

code.c

```c
// Praktikum EL3111 Arsitektur Sistem Komputer
// Modul : 1
// Percobaan : 1
// Tanggal : 23 September 2022
// Kelompok : 10
// Rombongan : B
// Nama (NIM) 1 : Ahmad Aziz (13220034)
// Nama (NIM) 2 : Gilbert Ng (13220032)
// Nama File : code.c
// Deskripsi : Demonstrasi proses kompilasi C
// Menjumlahkan deret bilangan sebanyak N_LOOP
#define N_LOOP 500
int main(void){
    int indeks;
    int accumulator;
    indeks = 0;
    accumulator = 0;
    while(indeks<N_LOOP){
        accumulator = accumulator + indeks;
        indeks = indeks + 1;
    }
    return accumulator;
}
```

code.i

```
# 0 "code.c"
# 0 "<built-in>"
# 0 "<command-line>"
# 1 "code.c"
# 13 "code.c"
int main(void){
    int indeks;
    int accumulator;
    indeks = 0;
    accumulator = 0;
    while(indeks<500){
        accumulator = accumulator + indeks;
        indeks = indeks + 1;
    }
    return accumulator;
}
```

code.s

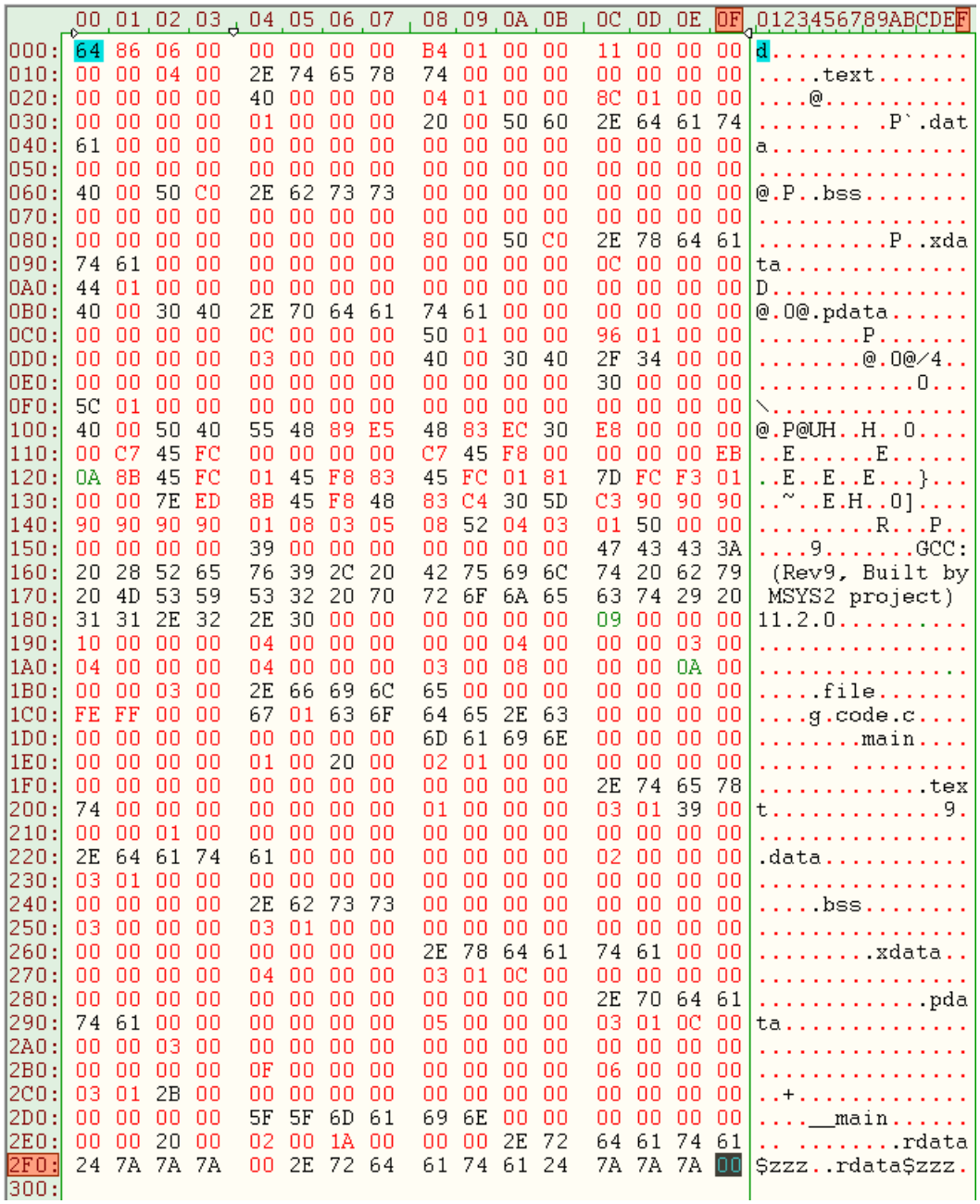
```
    .file   "code.c"
    .text
    .def    __main; .scl    2;  .type   32; .endef
```

```
    .globl  main
    .def    main;   .scl    2;  .type   32; .endef
    .seh_proc   main
main:
    pushq   %rbp
    .seh_pushreg    %rbp
    movq    %rsp, %rbp
    .seh_setframe   %rbp, 0
    subq    $48, %rsp
    .seh_stackalloc 48
    .seh_endprologue
    call    __main
    movl    $0, -4(%rbp)
    movl    $0, -8(%rbp)
    jmp .L2
.L3:
    movl    -4(%rbp), %eax
    addl    %eax, -8(%rbp)
    addl    $1, -4(%rbp)
.L2:
    cmpl    $499, -4(%rbp)
    jle .L3
    movl    -8(%rbp), %eax
    addq    $48, %rsp
    popq    %rbp
    ret
    .seh_endproc
    .ident  "GCC: (Rev9, Built by MSYS2 project) 11.2.0"
```

batch.bat

```
%~d0
cd "%~dp0"
gcc -E code.c > code.i
gcc -S code.c
gcc -c code.c
gcc -o code.exe code.c
code.exe
pause
```

code.o



```
      00 01 02 03  04 05 06 07  08 09 0A 0B  0C 0D 0E 0F  0123456789ABCDEF
000:  64 86 06 00  00 00 00 00  B4 01 00 00  11 00 00 00  d...............
010:  00 00 04 00  2E 74 65 78  74 00 00 00  00 00 00 00  .....text.......
020:  00 00 00 00  40 00 00 00  04 01 00 00  8C 01 00 00  ....@...........
030:  00 00 00 00  01 00 00 00  20 00 50 60  2E 64 61 74  ........ .P`.dat
040:  61 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  a...............
050:  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
060:  40 00 50 C0  2E 62 73 73  00 00 00 00  00 00 00 00  @.P..bss........
070:  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
080:  00 00 00 00  00 00 00 00  80 00 50 C0  2E 78 64 61  ..........P..xda
090:  74 61 00 00  00 00 00 00  00 00 00 00  0C 00 00 00  ta..............
0A0:  44 01 00 00  00 00 00 00  00 00 00 00  00 00 00 00  D...............
0B0:  40 00 30 40  2E 70 64 61  74 61 00 00  00 00 00 00  @.0@.pdata......
0C0:  00 00 00 00  0C 00 00 00  50 01 00 00  96 01 00 00  ........P.......
0D0:  00 00 00 00  03 00 00 00  40 00 30 40  2F 34 00 00  ........@.0@/4..
0E0:  00 00 00 00  00 00 00 00  00 00 00 00  30 00 00 00  ............0...
0F0:  5C 01 00 00  00 00 00 00  00 00 00 00  00 00 00 00  \...............
100:  40 00 50 40  55 48 89 E5  48 83 EC 30  E8 00 00 00  @.P@UH..H..0....
110:  00 C7 45 FC  00 00 00 00  C7 45 F8 00  00 00 00 EB  ..E......E......
120:  0A 8B 45 FC  01 45 F8 83  45 FC 01 81  7D FC F3 01  ..E..E..E...}...
130:  00 00 7E ED  8B 45 F8 48  83 C4 30 5D  C3 90 90 90  ..~..E.H..0]....
140:  90 90 90 90  01 08 03 05  08 52 04 03  01 50 00 00  .........R...P..
150:  00 00 00 00  39 00 00 00  00 00 00 00  47 43 43 3A  ....9.......GCC:
160:  20 28 52 65  76 39 2C 20  42 75 69 6C  74 20 62 79   (Rev9, Built by
170:  20 4D 53 59  53 32 20 70  72 6F 6A 65  63 74 29 20   MSYS2 project)
180:  31 31 2E 32  2E 30 00 00  00 00 00 00  09 00 00 00  11.2.0..........
190:  10 00 00 00  04 00 00 00  00 00 04 00  00 00 03 00  ................
1A0:  04 00 00 00  04 00 00 00  03 00 08 00  00 00 0A 00  ................
1B0:  00 00 03 00  2E 66 69 6C  65 00 00 00  00 00 00 00  .....file.......
1C0:  FE FF 00 00  67 01 63 6F  64 65 2E 63  00 00 00 00  ....g.code.c....
1D0:  00 00 00 00  00 00 00 00  6D 61 69 6E  00 00 00 00  ........main....
1E0:  00 00 00 00  01 00 20 00  02 01 00 00  00 00 00 00  ......  ........
1F0:  00 00 00 00  00 00 00 00  00 00 00 00  2E 74 65 78  .............tex
200:  74 00 00 00  00 00 00 00  01 00 00 00  03 01 39 00  t.............9.
210:  00 00 01 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
220:  2E 64 61 74  61 00 00 00  00 00 00 00  02 00 00 00  .data...........
230:  03 01 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
240:  00 00 00 00  2E 62 73 73  00 00 00 00  00 00 00 00  .....bss........
250:  03 00 00 00  03 01 00 00  00 00 00 00  00 00 00 00  ................
260:  00 00 00 00  00 00 00 00  2E 78 64 61  74 61 00 00  .........xdata..
270:  00 00 00 00  04 00 00 00  03 01 0C 00  00 00 00 00  ................
280:  00 00 00 00  00 00 00 00  00 00 00 00  2E 70 64 61  .............pda
290:  74 61 00 00  00 00 00 00  05 00 00 00  03 01 0C 00  ta..............
2A0:  00 00 03 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
2B0:  00 00 00 00  0F 00 00 00  00 00 00 00  06 00 00 00  ................
2C0:  03 01 2B 00  00 00 00 00  00 00 00 00  00 00 00 00  ..+.............
2D0:  00 00 00 00  5F 5F 6D 61  69 6E 00 00  00 00 00 00  ....__main......
2E0:  00 00 20 00  02 00 1A 00  00 00 2E 72  64 61 74 61  .. ........rdata
2F0:  24 7A 7A 7A  00 2E 72 64  61 74 61 24  7A 7A 7A 00  $zzz..rdata$zzz.
300:
```

1. Source code untuk tugas 3
disassembly_code_o.asm

```
code.o:     file format pe-x86-64


Disassembly of section .text:

0000000000000000 <main>:
   0: 55                    push   %rbp
   1: 48 89 e5              mov    %rsp,%rbp
   4: 48 83 ec 30           sub    $0x30,%rsp
   8: e8 00 00 00 00        call   d <main+0xd>
```

```
   d: c7 45 fc 00 00 00 00    movl   $0x0,-0x4(%rbp)
  14: c7 45 f8 00 00 00 00    movl   $0x0,-0x8(%rbp)
  1b: eb 0a                   jmp    27 <main+0x27>
  1d: 8b 45 fc                mov    -0x4(%rbp),%eax
  20: 01 45 f8                add    %eax,-0x8(%rbp)
  23: 83 45 fc 01             addl   $0x1,-0x4(%rbp)
  27: 81 7d fc f3 01 00 00    cmpl   $0x1f3,-0x4(%rbp)
  2e: 7e ed                   jle    1d <main+0x1d>
  30: 8b 45 f8                mov    -0x8(%rbp),%eax
  33: 48 83 c4 30             add    $0x30,%rsp
  37: 5d                      pop    %rbp
  38: c3                      ret
  39: 90                      nop
  3a: 90                      nop
  3b: 90                      nop
  3c: 90                      nop
  3d: 90                      nop
  3e: 90                      nop
  3f: 90                      nop
```

2. Tugas 4

batch.bat

```
%~d0
cd "%~dp0"
gcc -O0 -c code_O0.c
gcc -O1 -c code_O1.c
gcc -O2 -c code_O2.c
gcc -O3 -c code_O3.c
gcc -Os -c code_Os.c
gcc -Ofast -c code_Ofast.c
objdump -d code_O0.o > code_O0.s
objdump -d code_O1.o > code_O1.s
objdump -d code_O2.o > code_O2.s
objdump -d code_O3.o > code_O3.s
objdump -d code_Os.o > code_Os.s
objdump -d code_Ofast.o > code_Ofast.s
pause
```

code-O0.s

```
code_O0.o:     file format pe-x86-64


Disassembly of section .text:

0000000000000000 <main>:
```

```
   0: 55                    push   %rbp
   1: 48 89 e5              mov    %rsp,%rbp
   4: 48 83 ec 30           sub    $0x30,%rsp
   8: e8 00 00 00 00        call   d <main+0xd>
   d: c7 45 fc 00 00 00 00  movl   $0x0,-0x4(%rbp)
  14: c7 45 f8 00 00 00 00  movl   $0x0,-0x8(%rbp)
  1b: eb 0a                 jmp    27 <main+0x27>
  1d: 8b 45 fc              mov    -0x4(%rbp),%eax
  20: 01 45 f8              add    %eax,-0x8(%rbp)
  23: 83 45 fc 01           addl   $0x1,-0x4(%rbp)
  27: 81 7d fc f3 01 00 00  cmpl   $0x1f3,-0x4(%rbp)
  2e: 7e ed                 jle    1d <main+0x1d>
  30: 8b 45 f8              mov    -0x8(%rbp),%eax
  33: 48 83 c4 30           add    $0x30,%rsp
  37: 5d                    pop    %rbp
  38: c3                    ret
  39: 90                    nop
  3a: 90                    nop
  3b: 90                    nop
  3c: 90                    nop
  3d: 90                    nop
  3e: 90                    nop
  3f: 90                    nop
```

code-O1.s

```
code_O1.o:     file format pe-x86-64


Disassembly of section .text:

0000000000000000 <main>:
   0: 48 83 ec 28           sub    $0x28,%rsp
   4: e8 00 00 00 00        call   9 <main+0x9>
   9: b8 f4 01 00 00        mov    $0x1f4,%eax
   e: 83 e8 01              sub    $0x1,%eax
  11: 75 fb                 jne    e <main+0xe>
  13: b8 4e e7 01 00        mov    $0x1e74e,%eax
  18: 48 83 c4 28           add    $0x28,%rsp
  1c: c3                    ret
  1d: 90                    nop
  1e: 90                    nop
  1f: 90                    nop
```

code-O2.s

```
code_O2.o:     file format pe-x86-64
```

```
Disassembly of section .text.startup:

0000000000000000 <main>:
   0: 48 83 ec 28             sub    $0x28,%rsp
   4: e8 00 00 00 00          call   9 <main+0x9>
   9: b8 4e e7 01 00          mov    $0x1e74e,%eax
   e: 48 83 c4 28             add    $0x28,%rsp
  12: c3                      ret
  13: 90                      nop
  14: 90                      nop
  15: 90                      nop
  16: 90                      nop
  17: 90                      nop
  18: 90                      nop
  19: 90                      nop
  1a: 90                      nop
  1b: 90                      nop
  1c: 90                      nop
  1d: 90                      nop
  1e: 90                      nop
  1f: 90                      nop
```

code-O3.s

```
code_O3.o:     file format pe-x86-64


Disassembly of section .text.startup:

0000000000000000 <main>:
   0: 48 83 ec 28             sub    $0x28,%rsp
   4: e8 00 00 00 00          call   9 <main+0x9>
   9: b8 4e e7 01 00          mov    $0x1e74e,%eax
   e: 48 83 c4 28             add    $0x28,%rsp
  12: c3                      ret
  13: 90                      nop
  14: 90                      nop
  15: 90                      nop
  16: 90                      nop
  17: 90                      nop
  18: 90                      nop
  19: 90                      nop
  1a: 90                      nop
  1b: 90                      nop
  1c: 90                      nop
```

```
  1d: 90                        nop
  1e: 90                        nop
  1f: 90                        nop
```

code-Ofast.s

```
code_Ofast.o:     file format pe-x86-64


Disassembly of section .text.startup:

0000000000000000 <main>:
   0: 48 83 ec 28             sub    $0x28,%rsp
   4: e8 00 00 00 00          call   9 <main+0x9>
   9: b8 4e e7 01 00          mov    $0x1e74e,%eax
   e: 48 83 c4 28             add    $0x28,%rsp
  12: c3                      ret
  13: 90                      nop
  14: 90                      nop
  15: 90                      nop
  16: 90                      nop
  17: 90                      nop
  18: 90                      nop
  19: 90                      nop
  1a: 90                      nop
  1b: 90                      nop
  1c: 90                      nop
  1d: 90                      nop
  1e: 90                      nop
  1f: 90                      nop
```

code-Os.s

```
code_Os.o:     file format pe-x86-64


Disassembly of section .text.startup:

0000000000000000 <main>:
   0: 48 83 ec 28             sub    $0x28,%rsp
   4: e8 00 00 00 00          call   9 <main+0x9>
   9: b8 4e e7 01 00          mov    $0x1e74e,%eax
   e: 48 83 c4 28             add    $0x28,%rsp
  12: c3                      ret
  13: 90                      nop
```

```
  14: 90                    nop
  15: 90                    nop
  16: 90                    nop
  17: 90                    nop
  18: 90                    nop
  19: 90                    nop
  1a: 90                    nop
  1b: 90                    nop
  1c: 90                    nop
  1d: 90                    nop
  1e: 90                    nop
  1f: 90                    nop
```

3. Tugas 5

main_text.c

```c
// Praktikum EL3111 Arsitektur Sistem Komputer
// Modul : 1
// Percobaan : 5
// Tanggal : 23 September 2022
// Kelompok : 10
// Rombongan : B
// Nama (NIM) 1 : Ahmad Aziz (13220034)
// Nama (NIM) 2 : Gilbert Ng (13220032)
// Nama File : main_text.c
// Deskripsi : Demonstrasi MakeFile
// Memanggil prosedur test pada text.c
#include "text.h"
void main(void){
    test();
}
```

text.c

```c
// Praktikum EL3111 Arsitektur Sistem Komputer
// Modul : 1
// Percobaan : 5
// Tanggal : 23 September 2022
// Kelompok : 10
// Rombongan : B
// Nama (NIM) 1 : Ahmad Aziz (13220034)
// Nama (NIM) 2 : Gilbert Ng (13220032)
// Nama File : text.c
// Deskripsi : Demonstrasi MakeFile, Mencetak string ke layar
#include <stdio.h>
#include "text.h"
void test(void){
```

```
    printf("Arsitektur Sistem Komputer sangat menyenangkan!\n");
}
```

text.h

```
// Praktikum EL3111 Arsitektur Sistem Komputer
// Modul : 1
// Percobaan : 5
// Tanggal : 23 September 2022
// Kelompok : 10
// Rombongan : B
// Nama (NIM) 1 : Ahmad Aziz (13220034)
// Nama (NIM) 2 : Gilbert Ng (13220032)
// Nama File : text.h
// Deskripsi : Demonstrasi MakeFile, Mencetak string ke layar
#ifndef TES_H
    #define TES_H 100
    void test(void);
#endif
```

4. Tugas 6

makefile

```
all: main_text.exe
main_text.exe: main_text.o text.o
    gcc main_text.o text.o -o main_text.exe
main_text.o: main_text.c
    gcc -c main_text.c
text.o: text.c
    gcc -c text.c
```

5. Tugas 7

add.c

```
// Praktikum EL3111 Arsitektur Sistem Komputer
// Modul : 1
// Percobaan : 7
// Tanggal : 23 September 2022
// Kelompok : 10
// Rombongan : B
// Nama (NIM) 1 : Ahmad Aziz (13220034)
// Nama (NIM) 2 : Gilbert Ng (13220032)
// Nama File : add.c
// Deskripsi : Demonstrasi header file
// Menjumlahkan dua bilangan

#define START_VAL 0

int accum = START_VAL;
```

```
// Praktikum EL3111 Arsitektur Sistem Komputer
// Modul : 1
// Percobaan : 7
// Tanggal : 23 September 2022
// Kelompok : 10
// Rombongan : B
// Nama (NIM) 1 : Ahmad Aziz (13220034)
// Nama (NIM) 2 : Gilbert Ng (13220032)
// Nama File : add.h
// Deskripsi : Demonstrasi header file
// Header file fungsi add
#ifndef ADD_H
    int sum(int x, int y);
#endif
```

add.h

```
#ifndef ADD_H
    int sum(int x, int y);
#endif
```

main.c

```
// Praktikum EL3111 Arsitektur Sistem Komputer
// Modul : 1
// Percobaan : 7
// Tanggal : 23 September 2022
// Kelompok : 10
// Rombongan : B
// Nama (NIM) 1 : Ahmad Aziz (13220034)
// Nama (NIM) 2 : Gilbert Ng (13220032)
// Nama File : main.c
// Deskripsi : Demonstrasi header file
// Main program
#include <stdio.h>
#include "add.h"
int main(){
    int a;
    int b;
    scanf("%d %d", &a, &b);
    int c = sum(a,b);
    printf("Hasil penjumlahan %d + %d = %d",a,b,c);
    return 0;
}
```

6. Tugas 8

main.c

```
// Praktikum EL3111 Arsitektur Sistem Komputer
```

```c
// Modul : 1
// Percobaan : 8
// Tanggal : 23 September 2022
// Kelompok : 10
// Rombongan : B
// Nama (NIM) 1 : Ahmad Aziz (13220034)
// Nama (NIM) 2 : Gilbert Ng (13220032)
// Nama File : main.c
// Deskripsi : Demonstrasi procedure call dan stack
// Menghitung jumlah dari kuadrat bilangan

int square (int x){
    return x*x;
}

int squaresum (int y, int z){
    int temp1 = square(y);
    int temp2 = square(z);
    return temp1+temp2;
}

int main (void){
    int a = 5;
    int b = 9;
    return squaresum(a,b);
}
```

7. Tugass 9

fibo_main.c

```c
// Praktikum EL3111 Arsitektur Sistem Komputer
// Modul : 1
// Percobaan : 9
// Tanggal : 23 September 2022
// Kelompok : 10
// Rombongan : B
// Nama (NIM) 1 : Ahmad Aziz (13220034)
// Nama (NIM) 2 : Gilbert Ng (13220032)
// Nama File : fibo_main.c
// Deskripsi : Main file untuk fibonacci function
// Menghitung deret fibonacci sebanyak input user

#include "inputn.h"
#include "fibo.h"
#include <stdio.h>
int main(){
    int i;
    int N = inputn();
    for(i = 0; i < N; i++){
```

```
        printf("%d ", fibonacci(i));
    }
    return 0;
}
```

fibo.c

```
// Praktikum EL3111 Arsitektur Sistem Komputer
// Modul : 1
// Percobaan : 9
// Tanggal : 23 September 2022
// Kelompok : 10
// Rombongan : B
// Nama (NIM) 1 : Ahmad Aziz (13220034)
// Nama (NIM) 2 : Gilbert Ng (13220032)
// Nama File : fibo.h
// Deskripsi : Library fibonacci function
// Menghitung deret fibbonaci

#include "fibo.h"

int fibonacci(int n){
    if(n <= 1){
        return 1;
    }
    else{
        return fibonacci(n-1) + fibonacci(n-2);
    }
}
```

fibo.h

```
// Praktikum EL3111 Arsitektur Sistem Komputer
// Modul : 1
// Percobaan : 9
// Tanggal : 23 September 2022
// Kelompok : 10
// Rombongan : B
// Nama (NIM) 1 : Ahmad Aziz (13220034)
// Nama (NIM) 2 : Gilbert Ng (13220032)
// Nama File : inputn.h
// Deskripsi : Deklarasi fungsi fibonacci
// Header file library fibonacci

int fibonacci(int n);
```

inputn.c

```
// Praktikum EL3111 Arsitektur Sistem Komputer
```

```c
// Modul : 1
// Percobaan : 9
// Tanggal : 23 September 2022
// Kelompok : 10
// Rombongan : B
// Nama (NIM) 1 : Ahmad Aziz (13220034)
// Nama (NIM) 2 : Gilbert Ng (13220032)
// Nama File : inputn.c
// Deskripsi : Fungsi input dan validasi input user
// Fungsi input dan validasi input user

#include "inputn.h"
#include <stdio.h>

int inputn(){
    int n;
    printf("masukkan n: ");
    scanf("%d", &n);
    while (n<2){
        printf("masukkan n: ");
        scanf("%d", &n);
    }
    return n;
}
```

inputn.h

```c
// Praktikum EL3111 Arsitektur Sistem Komputer
// Modul : 1
// Percobaan : 9
// Tanggal : 23 September 2022
// Kelompok : 10
// Rombongan : B
// Nama (NIM) 1 : Ahmad Aziz (13220034)
// Nama (NIM) 2 : Gilbert Ng (13220032)
// Nama File : inputn.h
// Deskripsi : Deklarasi fungsi inputn
// Header file library inputn

int inputn();
```

makefile

```
all:
    gcc -o main fibo_main.c inputn.c fibo.c
clean:
    rm main
```