Program Studi Teknik Elektro ITB

Nama Kuliah (Kode) : Praktikum Arsitektur Sistem Komputer (EL3111)
Tahun / Semester : 2022-2023 / Ganjil
Modul : COMPILER BAHASA C DAN BAHASA ASSEMBLY INTEL® X86
Nama Asisten / NIM : _____
Nama Praktikan / NIM : Ahmad Aziz / 13220034

Tugas Pendahuluan

1. Jelaskan perbedaan antara masing-masing pilihan optimisasi dalam GCC (–O0, –O1, –O2, –O3, –Os, dan –Ofast)!

Optimisasi dalam kompilasi GCC adalah proses untuk membuat hasil kompilasi menjadi lebih optimal sesuai dengan fungsinya yaitu mengotimisasi, bisa dianalogikan seperti kita mengkompres suatu file. Melakukan optimisasi kompilasi dengan GCC dapat dilakukan dengan menambahkan perintah optimisasi yang disebut *flag* pada perintah kompilasi GCC.

Melakukan optimisasi yang artinya kita menginginkan suatu hal yang lebih dari hasil kompilasi tentunya kita juga harus mengorbankan sesuatu yang lain agar yang kita ingin kita optimisasi bisa didapatkan (adanya *trade-off*). Optimisasi pada GCC sendiri dapat melakukan beberapa jenis optimisasi yang dapat digunakan dengan *flag* yaitu -O0, -O1, -O2, -O3, -Os, dan -Ofast. Berbagai optimisasi tersebut berkaitan dengan parameter optimisasi sebagai berikut:

a. *execution speed*
b. *program size*
c. *compilation time*
d. *memory use*

Ketika kita memilih suatu opsi optimisasi, kita akan mendapatkan kelebihan dan kekurangan pada parameter-parameter diatas. Oleh karena itu pemilihan optimisasi ditentukan sesuai dengan kegunaan dan kebutuhan program yang dibuat.

Optimisasi O0 merupakan otimisasi yang akan mempercepat waktu kompilasi, O1 akan mengurasi waktu eksekusi dan ukuran kode, O2 optimisasi yang lebih tinggi dari O1 dengan trade-off yang lebih besar pula tentunya, begitu pula dengan O3. Os merupakan optimisasi untuk size kode, dan Ofast sama dengan O3 dan penambahan optimisasi fungsi matematis. Untuk lebih jelasnya berikut ini adalah tabel hubungan kelebihan dan kekurangan setiap parameter dengan jenis optimisasi kompilasi pada GCC:

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 sedikut.

Tabel hubungan parameter optimisasi dengan opsi optimisasi kompilasi GCC
Sumber: Modul Praktikum Arsikom

2. **Bahasa C merupakan bahasa yang banyak digunakan dalam membuat program pada beberapa platform. Sebagai contoh, bahasa C dapat digunakan untuk membuat program pada mikroprosesor berbasis Intel® x86. Bahasa C juga dapat digunakan untuk membuat program pada mikrokontroler AVR®. Di sisi lain, mikroprosesor Intel® x86 memiliki set instruksi yang jauh berbeda dibanding mikrokontroler AVR® ATmega. Menurut pengetahuan Anda tentang proses kompilasi bahasa C, apa yang menyebabkan bahasa C tetap dapat digunakan meskipun platform-nya berbeda?**

   Bahasa C merupakan bahasa pemrograman tingkat tinggi (beberapa referensi menyebut bahasa C pada level menegah) dimana merupakan bahasa yang dekat (dimengerti) dengan/oleh bahasa manusia. Pada dasarnya mesin tidak dapat membaca atau mengeksekusi langsung bahasa C, mesin hanya dapat membaca kode dalam bahasa mesin (binary). Oleh karena itu untuk dapat menjalankan bahasa C dibutuhkan kompiler.Kompiler akan melakukan proses penerjemahan bahasa C dalam beberapa tahapan yang disebut kompilasi untuk menghasilkan file yang dapat diesekusi (*executable*) dan dijalankan oleh mesin.

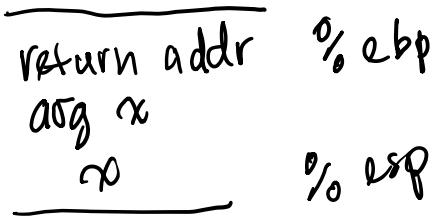
   Itulan mengapa bahasa C dapat dijalankan pada berbagai mesin. Karena itu bahasa pemrograman C bersifat *processor independent* yang artinya tidak tergantung oleh *processor* dimana program dijalankan. Bahasa C bahkan tidak membutuhkan mesin dengan arsitektur yang sama. Program yang ditulis dalam bahasa pemrograman C akan dikompilasi sesuai dengan sistem operasi dan hardware tempat program tersebut dijalankan.

3. **a. Pada file assembly tersebut, terdapat barisan kode assembly (ditampilkan di samping) yang selalu dieksekusi di awal sebuah prosedur. Apa fungsi kode-kode assembly tersebut?**

```
pushl %ebp
movl  %esp, %ebp
```

Kode pushl merupakan kode untuk melakukan “push” atau memasukkan base pointer register kedalam stack. Nilai dari register %ebp akan dicopy ke lokasi address pada stack pointer. Kemudian kode “movl” akan memindahkan stack pointer yang sebelunya ke register base pointer yang baru. Sehingga, kode tersebut memasukkan return value prosedur tersebut ke dalam stack baru untuk nantinya dapat menyimpan data saat fungsi dijalankan dan akan di “pop” ketika sudah mengembalikan return atau selesai dijalankan.

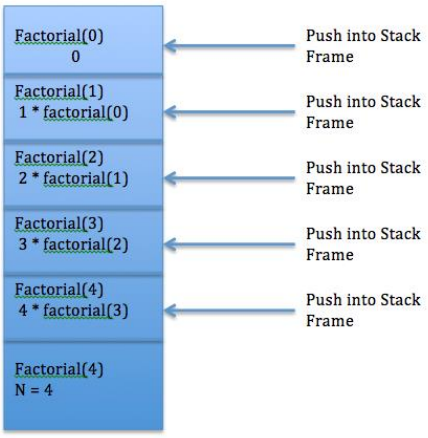
b. Gambarkan isi stack sebelum instruksi (imull 8(%ebp),%eax) pada saat prosedur square dipanggil pertama kali oleh prosedur squaresum! (isi stack saat main memanggil squaresum tidak perlu digambarkan)



c. Prosedur rekursif merupakan prosedur yang memanggil dirinya sendiri secara berulang-ulang hingga kondisi berhenti dipenuhi. Berdasarkan pengetahuan Anda tentang procedure call dan stack ini, bagaimanakah penggunaan memory pada prosedur rekursif?

Prosedur rekursif adalah prosedur yang akan terus menerus memanggil dirinya sendiri hingga dia mencapai base conditionnya. Pada dasarnya stack prosedur rekursif sama saja dengan prosedur atau fungsi biasa. Ketika fungsi rekursif sedang dieksekusi untuk pertama kali, stack frame fungsi tersebut akan dibuat. Setelah itu fungsi rekursi tersebut akan memanggil dirinya sendiri, namun dengan

parameter yang berbeda tentunya (jika fungsi rekursinya benar). Maka akan dibuat stack frame baru untuk fungsi tersebut, begitu seterusnya hingga mencapat base condition dari rekursifnya.Berikut ini ilustrasi stack frame pada fungsi rekursi factorial sederhana:



Gambar diagram ilustrasi stack frame pada fungsi rekursif
Source: http://knowledge-cess.com/tag/recursive-analysis/

4. Apa itu stack dalam arsitektur sistem komputer? Jelaskan fungsinya!

Stack adalah mekanisme penggunaan memori dalam suatu komputer. Stack memiliki lokasi atau suatu blok dalam memori dalam konfigurasi/aturan LIFO (*last in first out*). Stack ini berfungsi untuk menyimpan data sementara dari sebuah fungsi atau prosedur pada sebuah program.

5. Jelaskan apa saja yang terjadi pada stack ketika fungsi memanggil fungsi-fungsi yang lain!

Ketika dalam suatu fungsi terjadi pemanggilan fungsi lain di dalamnya, maka akan dibuat stack frame baru sesuai dengan mekanisme alokasi stack yaitu LIFO. Pada stack frame baru tersebut, data local variable, address dan argumen fungsi akan di store kedalam frame tersebut dan register stack pointer dan base pointernya bergeser ke stack frame baru tersebut. Setelah fungsi tersebut selesai dijalankan atau sudah mereturn nilainya, maka stack frame tersebut akan “pops”/“dihapus” (tidak benar-benar dihapus) dan stack frame akan kembali ke fungsi sebelumnya yang memanggil fungsi tersebut. Stack pointer dan base pointernya juga akan kembali pada stack frame fungsi awal.

6. Gambarkan stack pada keadaan nomor 5!

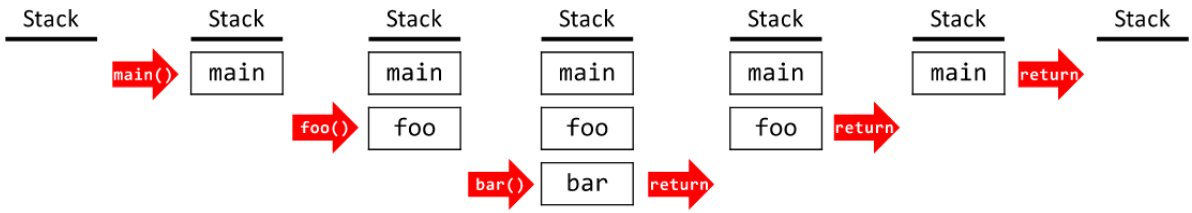
Seperti yang sudah dijelaskan pada jawaban sebelumnya, jika sebuah fungsi memanggil fungsi lain, maka akan dibuat stack baru diatas stack fungsi yang sudah ada. Berikut ini adalah contoh sederhana kode dan gambaran stacknya.

```
void bar() {
    // some code
}

void foo() {
    bar();
}

int main() {
    foo();
}
```

Pada kode dengan pemanggilan fungsi seperti diatas maka stack akan bertambah ketika fungsi dipanggil dan akan dihapus ketika fungsi tersebut mereturn nilai atau selesai dieskekusi seperti ilustrasi pada gambar berikut ini:



Gambar ilustrasi stack frame
Source: https://eecs280staff.github.io/notes/02_ProceduralAbstraction_Testing.html