

**Tugas Besar**  
**IF2230 - Sistem Operasi**  
**Milestone 01 of ??**

*"Buset keos."*

**Pembuatan Sistem Operasi Sederhana**  
**Booting, Kernel, System Call**

Dipersiapkan oleh :  
Asisten Lab Sistem Terdistribusi

Didukung Oleh :



**Waktu Mulai :**  
Rabu, 16 Februari 2022, 20.22 WIB

**Waktu Akhir :**  
Selasa, 1 Maret 2022, 23.59 WIB

# I. Latar Belakang

Saat ini, Anda merupakan mahasiswa semester 4 dari perguruan tinggi yang terkenal sangat prestisius. Saat Anda memasuki perguruan tinggi ini, Anda memiliki banyak mimpi untuk dilakukan, beberapa di antaranya yaitu: mendapatkan IP minimal 3.8 dan maksimal 5, aktif berorganisasi maupun berhimpun, magang di suatu *startup hectocorn* (*startup* dengan nilai valuasi yang bisa disetarakan dengan Facebook, Google, Amazon), berhenti menjadi wibu, dan mempunyai pacar. Dua mimpi pertama yang disebutkan telah berhasil Anda raih, kini saatnya menggapai mimpi ketiga, magang.



*Kaiba Corp, startup hectocorn*

Untuk itu Anda mulai menjelajahi platform digital dimana banyak pekerjaan ditawarkan yang bernama *LinkedOut*, tampak sebuah *startup hectocorn* yang menarik perhatian Anda, *Kaiba Corp*. Sebuah startup tersohor yang berfokus kepada dunia entertainment dan gaming industry. Untuk lebih spesifiknya startup ini bertujuan untuk membuat teknologi yang berkaitan dengan *Duel Monsters* game, yang saat ini mereka sedang gencar dalam pengembangan *dueling disk*.

Pada laman *LinkedOut* startup tersebut, Anda melihat sebuah lowongan kerja sebagai *Operating System Developer Intern* dengan spesifikasi sebagai berikut:

1. Pengalaman minimal 5+ tahun dalam sistem operasi
2. Pengetahuan yang kuat mengenai sistem operasi secara umum dan memahami konsep syscall dengan baik
3. Mampu membuat OS

4. Kemampuan yang bagus dengan bahasa C, C++, C#, C-, C\*, C&, C%, C!!!1111!!!!11
5. Menggunakan bahasa assembly sebagai bahasa komunikasi sehari-hari
6. Bisa memperbaiki kulkas juga diperlukan
7. Mampu melakukan proses instalasi ubuntu bajakan (**penting!**)
8. Paham dan mengerti cara bermain YuGiOh
9. Mampu mendapatkan 5 bagian Exodia dalam turn pertama
10. Mampu bekerja di bawah tekanan tubes, tucil, tused (tubes sedang), quiz, uts, uts 2, uas, cobaan hidup
11. Mampu bekerja 34 jam sehari serta berpikir dengan kepala dingin dalam menyelesaikan setiap cobaan yang ada.

Tentu saja Anda yang telah ditempa dengan sangat keras di itebe merasa sangat yakin dalam memenuhi *requirements* pada pekerjaan tersebut. Dengan cepat Anda mulai mempersiapkan CV sesuai dengan spesifikasi yang dibutuhkan dan mengikuti arahan dari seminar tentang membuat CV yang telah Anda ikuti sejak semester pertama.



Seto Kaiba, CEO dari Kaiba Corp



Seto Mulyadi, Psikolog anak Indonesia

Beberapa jam setelah mengirimkan CV tersebut terdapat notifikasi email dari HR startup tersebut untuk melakukan interview pertama. Singkat cerita setelah selesai menjalani interview, Anda mendapatkan pemberitahuan dari HR yang menyatakan bahwa CEO Kaiba Corp sangat tertarik dan menginginkan Anda untuk bekerja langsung padanya karena kemampuan yang Anda miliki. Tetapi sebelum itu akan dilakukan *technical interview* dengan Seto Kaiba, CEO Kaiba Corp.

*Technical interview* dimulai, Anda diberikan link suatu meet online (karena lagi covid, interviewnya online gan) dalam meet tersebut Anda melihat banyak calon pemegang lain yang masuk ke dalam meet yang sama, tampaknya mereka merupakan peserta yang lolos interview pertama dan memiliki kemampuan yang dibutuhkan oleh CEO tersebut dan banyak diantaranya merupakan teman sejurusan Anda. Tanpa berbasah basi Kaiba meminta calon

pemagang untuk melakukan *share screen* dan mulai membuka code editor. Ia meminta kandidat magang untuk membuat sistem operasi dalam waktu 14 jam, barang siapa yang dapat melakukannya, akan diterima dan ditempatkan sebagai tim khusus yang bekerja langsung pada Kaiba. Dibawah ini merupakan spesifikasi dari sistem operasi yang diinginkannya, mampukah Anda menyelesaikan sistem operasi tersebut?

## II. Deskripsi Tugas

Pada interview ini, kalian akan membuat sebuah sistem menakjubkan bernama sistem operasi. Sistem operasi yang akan kalian buat akan berjalan pada sistem 16-bit yang nanti akan disimulasikan dengan *emulator* bochs. Tugas ini akan dibagi menjadi beberapa *milestone* dan inilah yang akan dikerjakan di *milestone* pertama

- Menyiapkan *disk image*
- Melakukan instalasi *bootloader*
- Implementasi kernel sederhana
- Menjalankan sistem operasi
- Melakukan implementasi beberapa *syscall*
  - Membaca *input* dari keyboard
  - Menulis *output* ke layar
  - Membersihkan (*clear*) layar
- Pembuatan *script* untuk otomatisasi proses *build OS*
- Memahami cara kerja *interrupt*
- Memahami cara kerja *kernel.asm*

## III. Langkah Pengerjaan

### 3.1. Menyiapkan alat dan repository

Semua instruksi berikut ini akan mengasumsikan Anda menjalankan Linux. Untuk program-program yang sudah harus terinstal pada sistem operasi Anda adalah sebagai berikut:

- *Netwide assembler* (<https://www.nasm.us/>) untuk kompilasi program assembly
- *Bruce C Compiler* (<https://linux.die.net/man/1/bcc>) untuk kompilasi C 16 bit (GCC sudah tidak mendukung 16 bit yang murni)
- *ld86* (<https://linux.die.net/man/1/ld86>) untuk melakukan *linking object code*
- *Bochs* (<http://bochs.sourceforge.net/>) sebagai emulator untuk menjalankan sistem operasi

**Sangat dianjurkan untuk memakai sistem operasi Ubuntu 20.04.** Pada Ubuntu 20.04 berikut adalah perintah yang dapat digunakan untuk menginstal program-program di atas.

```
sudo apt update
sudo apt install nasm bcc bin86 bochs bochs-x make
```

Selanjutnya lakukan *login akun GitHub* dan akses *GitHub Classroom* menggunakan tautan yang akan diberikan oleh asisten. Pada *GitHub Classroom* akan terdapat *kit template* yang dapat digunakan untuk membuat dasar repository tugas besar ini.

### 3.2. Persiapan *disk image*

Sebelum memulai, Anda membutuhkan sebuah *image* yang akan digunakan untuk menyimpan sistem operasi Anda. Untuk tugas ini akan dibuat sebuah *image* disket berukuran 1.44 megabyte. Untuk membuatnya bisa digunakan *command line utility* dd. Contoh berikut mengasumsikan terdapat folder output bernama *out*.

```
dd if=/dev/zero of=out/system.img bs=512 count=2880
```

### 3.3. *Bootloader*

Saat komputer pertama kali melakukan *booting*, BIOS pada komputer akan mencari program untuk dijalankan. Program yang pertama kali dijalankan ini biasa disebut *bootloader* dan tugasnya adalah melakukan *booting* ke sistem operasi.

Untuk tugas ini, file *bootloader.asm* sudah disediakan dan tinggal dikompilasi dengan menggunakan *nasm*

```
nasm src/asm/bootloader.asm -o out/bootloader
```

Kemudian bootloader dapat dimasukkan ke dalam *disk image* dengan perintah

```
dd if=out/bootloader of=out/system.img bs=512 count=1 conv=notrunc
```

### 3.4. Pembuatan Kernel

Seperti sudah dijelaskan sebelumnya kode *bootloader* adalah kode yang pertama kali dijalankan oleh BIOS. Namun kode tersebut sangatlah kecil sehingga tidak mungkin memuat seluruh sistem operasi. Oleh karena itu, terdapat sebuah kernel terpisah yang nantinya akan dijalankan oleh *bootloader*.

Desain kernel untuk tugas besar ini dibuat sesederhana mungkin agar dapat lebih mudah dimengerti. Jika dilihat, terdapat file *kernel.asm* yang merupakan kode *assembly* dari kernel. Tugas anda adalah membuat file *kernel.c* yang merupakan kode utama kernel. Beberapa fungsi sudah disediakan oleh kode *kernel.asm* untuk digunakan pada file *kernel.c*. Jika dilihat pada bagian atas *kernel.asm* terdapat baris seperti berikut

```
global _putInMemory
```

Artinya, file *kernel.asm* mengimplementasi fungsi `putInMemory()` yang nantinya bisa Anda gunakan. Daftar fungsi-fungsi tersebut beserta *signature*-nya adalah

- `void putInMemory(int segment, int address, byte b)`  
Fungsi ini menulis sebuah nilai byte pada *segment* memori dengan *offset* tertentu. Lokasi tujuan adalah  $(segment * 0x10 + address)$ .
- `int interrupt(int number, int AX, int BX, int CX, int DX)`  
Fungsi ini memanggil sebuah *interrupt* dengan nomor tertentu dan juga meneruskan parameter **AX**, **BX**, **CX**, **DX** berukuran 16-bit sebagai register yang bernama dan berukuran sama ke *interrupt* tersebut. Perhatikan bahwa fungsi ini mengembalikan register **AL** yang telah dirubah oleh *interrupt* yang dipanggil.
- `void makeInterrupt21()`  
Fungsi ini mempersiapkan tabel *interrupt vector table* untuk memanggil *interrupt handler* yang dinamai **handleInterrupt21** jika *interrupt 0x21* terpanggil.
- `void handleInterrupt21(int AX, int BX, int CX, int DX)`  
Berbeda dengan ketiga fungsi diatas yang telah diimplementasikan pada *assembly*, fungsi ini akan diimplementasikan menggunakan bahasa C. Fungsi ini dipanggil saat terjadi *interrupt* nomor 0x21.

```
int main() {  
    char *string = "Hai"; // Deklarasikan variabel pada awal scope  
    int i = 0;
```

```

for (i = 0; i < 3; i++) {
    byte warna = 0xD;
    putInMemory(0xB000, 0x8000 + 2*i, string[i]);
    putInMemory(0xB000, 0x8001 + 2*i, warna);
}

while (true);
}

void handleInterrupt21(int AX, int BX, int CX, int DX) {
    // Definisi kosong
}

```

Kode di atas akan menghasilkan tulisan “Hai” di pojok kiri atas layar. Perhatikan bahwa:

- Untuk membantu memperjelas kode telah didefinisikan dua tipe data sederhana yang dapat digunakan yaitu **byte** dan **bool** yang terletak pada header *std\_type.h*
- 0xB000 adalah offset memori yang mendefinisikan karakter pada posisi (0, 0), dimana (0, 0) adalah posisi kiri atas layar.
- Nilai 0xB000 tidak bisa direpresentasikan pada satu 16 bit integer. Untuk mendapatkan nilai tersebut digunakan *segment*. Nilai segment dari *video memory* adalah 0xB000 dan address bernilai 0x8000 sehingga menghasilkan  $0xB000 * 0x10 + 0x8000 = 0xB8000$ .
- 0x8001 adalah offset memori yang mendefinisikan warna dari karakter pada posisi (0, 0).
- Rumus umum untuk posisi alamat memori karakter adalah  $0x8000 + (80 * y + x) * 2$
- Rumus umum untuk posisi alamat memori warna adalah  $0x8001 + (80 * y + x) * 2$
- **while (true)** digunakan agar kernel tidak reboot.
- **handleInterrupt21()** harus terdefinisi karena dideklarasikan sebagai **extern** di *kernel.asm* (seperti di mesin karakter Alstrukdat). Fungsi ini akan diisi nanti.

Setelah itu *kernel* sudah dapat di-*compile* dengan perintah berikut

```

bcc -ansi -c -o out/kernel.o src/c/kernel.c
nasm -f as86 src/asm/kernel.asm -o out/kernel_asm.o
ld86 -o out/kernel -d out/kernel.o out/kernel_asm.o
dd if=out/kernel of=out/system.img bs=512 conv=notrunc seek=1

```

Arti dari perintah di atas:

- bcc akan mengkompilasi kernel.c menjadi *object code* kernel.o
- nasm akan mengkompilasi kernel.asm menjadi *object code* kernel\_asm.o
- Kedua *object code* tersebut bisa dianggap potongan dari kode utuh kernel yang akan digabungkan dalam proses bernama *linking*. Perintah ld86 akan digunakan untuk tugas ini.



- dd akan digunakan kembali untuk memasukkan kernel ke *disk image* di sektor 1.

### 3.4. Menjalankan Sistem Operasi

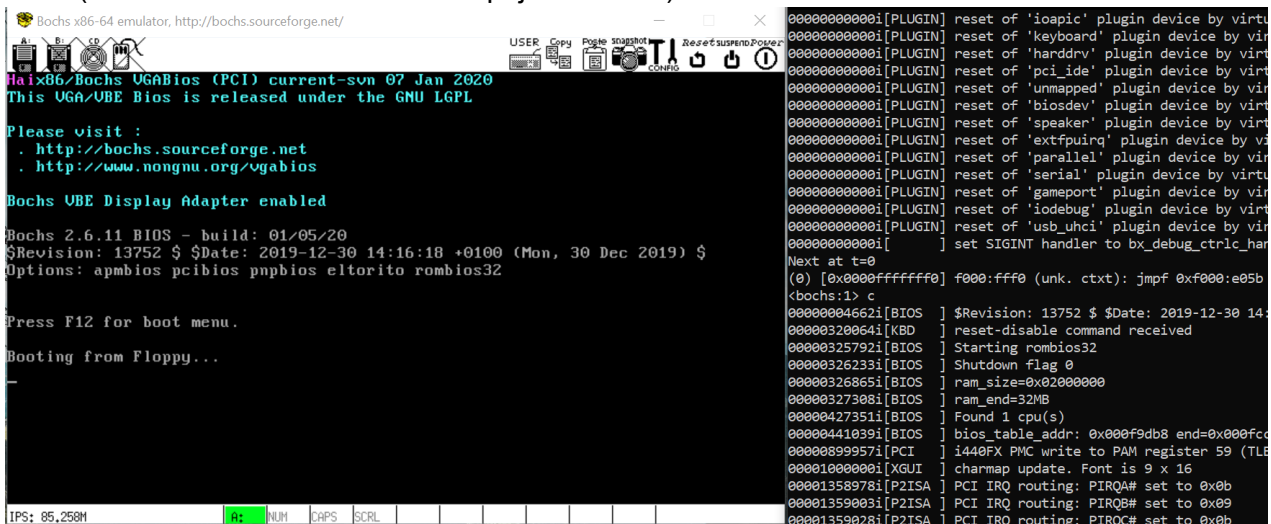
Untuk menjalankan sistem operasi yang Anda buat, akan digunakan emulator bochs. Untuk menjalankannya dibutuhkan sebuah file konfigurasi yang sudah diberikan dengan nama *if2230.config*. Jika menggunakan WSL, pastikan aplikasi X server pada Windows telah berjalan dan terkonfigurasi dengan benar

```
bochs -f src/config/if2230.config
```

Setelah itu akan muncul sebuah *prompt* pada terminal. Ketik 'c' (artinya *continue*) dan enter pada *prompt* tersebut.

```
00000000000i[ ] reading configuration from if2230.config
00000000000e[ ] if2230.config:195: 'vga_update_interval' will be replaced by new 'vga: update_freq' option.
00000000000e[ ] if2230.config:204: 'keyboard_serial_delay' will be replaced by new 'keyboard' option.
00000000000e[ ] if2230.config:221: 'keyboard_paste_delay' will be replaced by new 'keyboard' option.
00000000000e[ ] if2230.config:257: 'i440fxsupport' will be replaced by new 'pci' option.
00000000000i[ ] lt_dlhandle is 0x555f4b8023a0
00000000000i[PLGIN] loaded plugin libbx_x.so
00000000000i[ ] installing x module as the Bochs GUI
00000000000i[ ] using log file bochsout.txt
Next at t=0
(0) [0x00000000fffff0] f000:fff0 (unk. ctxt): jmp far f000:e05b      ; ea5be00f0
<bochs:1> c
[ ]
```

Berikutnya akan muncul sebuah *window* baru yang akan langsung menjalankan sistem operasi Anda (bisa dilihat dari tulisan "Hai" di pojok kiri atas)



### 3.5. Implementasi *interrupt 0x21*

Seperti pada semua sistem operasi, sistem operasi yang Anda buat juga harus mempunyai *syscall*. Pada sistem operasi ini *syscall* diimplementasikan dengan menggunakan *interrupt 0x21*. Tambahkan kode berikut pada fungsi `handleInterrupt21()` dan `main()`

```
int main() {
```

```

char buf[128];
clearScreen();
makeInterrupt21();

printString("Halo dunia!\r\n");
readString(buf);
printString(buf);

while (true);
}

void handleInterrupt21(int AX, int BX, int CX, int DX) {
    switch (AX) {
        case 0x0:
            printString(BX);
            break;
        case 0x1:
            readString(BX);
            break;
        default:
            printString("Invalid interrupt");
    }
}

```

Pada file *kernel.h* telah dideklarasikan beberapa fungsi

```

void printString(char *string);
void readString(char *string);
void clearScreen();

```

Tugas Anda adalah melengkapi implementasi tiap fungsi yang ada.

### 3.5.1. Implementasi printString dan readString

Kedua fungsi ini sangatlah sederhana dan mudah diimplementasikan. Dalam implementasinya hanya perlu menggunakan fasilitas yang disediakan oleh BIOS yang dapat dipanggil via *interrupt*. *Interrupt* yang dapat digunakan adalah *interrupt* 0x10 (tuliskan, *teletype output*) dan 0x16 (baca). Untuk penggunaannya dapat melihat referensi yang diberikan.

Spesifikasi untuk kedua fungsi adalah sebagai berikut

- **printString()** dapat menampilkan *null terminated string* dengan *character set* ASCII pada lokasi kursor.
- **readString()** dapat menerima input *non-control character* ASCII dan memasukkannya pada buffer. **readString** berhenti membaca ketika mendapatkan *carriage return* / tombol *enter*.

Untuk *behaviour* ketika **readString()** menerima *control character* seperti *backspace* dibebaskan.

### 3.5.2. Implementasi clearScreen

Fungsi ini digunakan untuk menghapus layar. Implementasi dapat menggunakan manipulasi memori dengan `putInMemory()` atau menggunakan `interrupt 0x10` (video mode). Mode default yang digunakan adalah *video mode 3* yang memiliki ukuran 80x25 karakter tidak termasuk memori untuk warna.

Spesifikasi untuk `clearScreen()` adalah sebagai berikut

- Menghapus layar
- Memindahkan kursor ke pojok kiri atas
- Mengganti buffer warna menjadi warna putih / 0x7 atau 0xF

### 3.6. Pembuatan Script

Untuk mempermudah proses *development*, diperlukan script untuk melakukan otomatisasi serangkaian proses *build* sistem operasi. Implementasi script dibebaskan kepada *interviewee* tetapi dianjurkan untuk menggunakan dan melengkapi makefile yang disediakan dalam kit. Berikut adalah resep-resep yang ada pada makefile

```
all: diskimage bootloader stdlib kernel

diskimage:
    # Pembuatan image

bootloader:
    # Pembuatan bootloader

kernel:
    # Pembuatan kernel

stdlib:
    # Opsional

run:
    bochs -f src/config/ief2230.config

build-run: all run
```

Dianjurkan untuk mengimplementasikan `std_lib.c` agar memudahkan operasi yang sering dilakukan.

## IV. Penilaian

### 1. Initialization and setup

- Penyiapan dan menjalankan sistem operasi dasar (20)

## **2. Basic teletype operation**

- *printString* berjalan dengan baik (20)
- *readString* berjalan dengan baik (20)
- *clearScreen* berjalan dengan baik (20)

## **3. Automation and build script**

- Sistem operasi dapat dijalankan dengan *make build-run* (20)

## V. Pengumpulan dan Deliverables

1. Untuk tugas ini Anda diwajibkan menggunakan *version control system* **git** dengan menggunakan sebuah *repository* **private** di Github Classroom “**Lab Sister 20**” (gunakan surel *student* agar gratis). Invitation ke dalam Github Classroom “**Lab Sister 20**” akan diberikan pada hari Senin, 21 Februari 2022.
2. Kit untuk semua milestone tugas besar IF2230 tersedia pada repository GitHub berikut [link repository](#).
3. Kreativitas dalam pengerjaan sangat dianjurkan untuk memperdalam pemahaman. Penilaian sepenuhnya didasarkan dari [kriteria penilaian](#), bukan detail implementasi.
4. Pada tugas besar ini akan digunakan bahasa *ANSI C* dan *nasm x86*. Disarankan untuk membuat mayoritas sistem operasi menggunakan *ANSI C* tetapi juga diperbolehkan untuk membuat kode *assembly* sendiri.
5. Setiap kelompok diwajibkan untuk membuat tim dalam Github Classroom dengan **nama yang sama pada spreadsheet kelompok**. Mohon diperhatikan lagi cara penamaan kelompok agar bisa sama dengan nama repository Anda nanti.
6. Meskipun commit tidak dinilai, lakukanlah *commit* yang wajar dan sesuai *best practice* (tidak semua kode satu *commit*).
7. File yang harus terdapat pada *repository* adalah file-file *source code* dan *script* (jika ada) sedemikian rupa sehingga jika diunduh dari github dapat dijalankan. Dihimbau untuk tidak memasukkan *binary* dan *temporary files* hasil kompilasi ke *repository* (manfaatkan **.gitignore**).
8. Isikan nama kelompok dan anggotanya pada [link berikut](#), paling lambat tanggal 19 Februari 2022 22.30 WIB.
9. **Mulai** Rabu, 16 Februari 2022, 20.22 WIB waktu server.  
**Deadline** Selasa, 1 Maret 2022, 23.59 WIB waktu server.  
Perubahan kode di luar waktu nilai dan *deadline* akan dikenakan pengurangan nilai.
10. Pengumpulan dilakukan dengan membuat **release** dengan tag **v.1.0.0** pada repository yang telah kelompok Anda buat sebelum deadline. Pastikan tag sesuai format.  
**Repository team yang tidak memiliki tag ini akan dianggap tidak mengumpulkan Milestone 1.**
11. Kami akan **menindaklanjuti segala bentuk kecurangan** yang terstruktur, masif, dan / atau sistematis.
12. Diharapkan untuk mencoba mengerjakan tugas besar ini terlebih dahulu sebelum mencari sumber inspirasi dari *google*, *repository*, atau teman yang sudah selesai. Namun **dilarang melakukan copy-paste langsung** kode orang lain (selain

spesifikasi). *Copy-paste* kode secara langsung akan dianggap melakukan kecurangan.

13. Dilarang melakukan kecurangan lain yang merugikan peserta mata kuliah IF2230.
14. Jika ada pertanyaan atau masalah pengerjaan harap segera menggunakan sheets QnA mata kuliah IF2230 Sistem Operasi pada [link berikut](#).
15. Sekali lagi, **sangat dianjurkan untuk memakai sistem operasi Ubuntu 20.04.**

## VI. Tips dan Catatan

1. Beberapa tips jika jarang menggunakan bahasa C : ANSI C tidak memiliki support [variable length array](#), deklarasikan semua array dengan ukuran konstan. C memperbolehkan pemanggilan yang lebih dari parameter (ex. `int main()` dapat dipanggil dengan `main(1, 1, 2, 3, 5)`). Tidak ada tipe data *string*, *string* pada C direpresentasikan sebagai [null-terminated string](#) pada *array of char*. C dan sebagian besar bahasa turunannya menggunakan [short-circuit evaluation](#) untuk operasi logika, hal ini menyebabkan sangat tidak direkomendasikan untuk memanggil fungsi yang memiliki *side effect* pada *ekspresi kondisional* (ex. `if (true || printString("hehe"))`), teks tersebut tidak akan ter-print). Anggap operator `sizeof` sebagai *compile time operator*, bukan sebagai *runtime operator* (ex. `sizeof(arr)` akan mengembalikan ukuran konstan maksimum *array*, bukan panjang yang terisi).
2. bcc tidak menyediakan *check* sebanyak gcc sehingga ada kemungkinan kode yang Anda buat berhasil *compile* tapi *error*. Untuk mengecek bisa mengcompile dahulu dengan gcc dan melihat apakah *error*.
3. Pastikan flag -d selalu ada ketika menjalankan perintah ld86. Flag tersebut membuat ld86 menghapus header pada output file.
4. Konsekuensi dari flag -d atas adalah **urutan definisi fungsi berpengaruh terhadap program**. Pastikan definisi fungsi pertama kali merupakan `int main()`. Deklarasikan fungsi selain `main()` pada header atau bagian atas program jika ingin memanggil fungsi lain pada `main()`.
5. Compiler bcc dengan opsi ANSI C hanya memperbolehkan deklarasi variabel pada awal scope. Deklarasi variabel pada tengah kode akan mengakibatkan error.
6. Spesifikasi **tidak** mewajibkan untuk *menghandle* seluruh *edge case* dan *abuse case*. Namun *jika memiliki waktu tambahan*, direkomendasikan untuk menambah *handler*.
7. Untuk melihat isi dari *disk* bisa digunakan utilitas *hexedit* untuk Linux dan *HxD* untuk Windows.
8. Walaupun kerapihan tidak dinilai langsung, kode yang rapi akan sangat membantu saat *debugging*.
9. Fungsi-fungsi dari *stdc* yang biasa Anda gunakan seperti **strlen** dan lainnya tidak tersedia di sini. Jika anda mau menggunakannya, anda harus membuatnya sendiri. Direkomendasikan untuk melengkapi definisi operasi-operasi umum pada *std\_lib.c*
10. Tahap *debugging* secara *dynamic* dapat menggunakan debugger yang disediakan *bochs* (memiliki fitur *gdb* dasar), menggunakan kondisional `if (a == b) printString("ok\n");` untuk *sanity check*, dan mengimplementasikan `printf()` jika diperlukan. Gunakan *hex editor* untuk melakukan *debugging* pada *storage*.

## VII. Referensi Tambahan

### 7.1. x86 Real Mode

Sebuah *processor* x86 mempunyai beberapa mode, yang paling awal ada yaitu *real mode* yang paling simpel. Karakteristik dari mode ini adalah ukuran alamat memori sepanjang 20-bit (artinya hanya sekitar 1 *Mebibyte* data yang dapat dialamatkan). Selain itu pada mode ini tidak terdapat konsep proteksi memori sehingga jika tidak diatur dengan baik **program dapat saling menggunakan blok memori yang sama** (memori yang digunakan program melebihi tempat yang dialokasikan sehingga memakai blok memori program lain) sehingga menghasilkan *error random*. Untuk lebih detail bisa melihat referensi pada [https://wiki.osdev.org/Real\\_Mode#Information](https://wiki.osdev.org/Real_Mode#Information)

Selain mode *real* terdapat juga mode *protected* yang digunakan oleh semua sistem operasi modern. Namun mode ini jauh lebih kompleks dibanding mode *real* karena pada mode ini sistem operasi harus mengatur banyak hal sendiri seperti *global descriptor table* untuk mengatur segmen memori.

Seperti prosesor, tampilan grafis juga memiliki beberapa mode, salah satunya adalah mode teks. Pada mode ini (yang biasa disebut juga mode VGA 3) diperbolehkan penulisan langsung ke memori *text buffer* yang terletak pada alamat **0xB8000** ke atas. Mode ini menyediakan ukuran layar 80x25 karakter dengan dukungan untuk warna. Penulisan data ke alamat memori *text buffer* akan langsung ditampilkan di layar. Untuk lebih detail bisa melihat referensi pada [https://wiki.osdev.org/Text\\_UI](https://wiki.osdev.org/Text_UI)

Jika dilihat pada potongan kode di atas, fungsi `putInMemory` menerima *segment* dan *offset* yang akan dikalkulasikan dengan rumus  $(\text{segment} \ll 4) + \text{offset}$  atau  $\text{segment} * 0x1000 + \text{offset}$ . Hal ini diperlukan untuk mengatasi ukuran register yang hanya 16 bit sedangkan alamat memori berukuran 20 bit. Untuk lebih detail bisa melihat referensi pada [https://wiki.osdev.org/Real\\_Mode#Memory\\_Addresssing](https://wiki.osdev.org/Real_Mode#Memory_Addresssing)

### 7.2. BIOS Interrupt Services

*BIOS interrupt calls* adalah fasilitas yang diberikan hardware untuk memanggil *Basic Input/Output System* (BIOS) software pada komputer. BIOS interrupt calls dapat melakukan kontrol langsung pada hardware atau fungsi I/O yang diminta oleh program atau mengembalikan informasi terkait sistem. Umumnya pada bahasan terkait *interrupt*, secara implisit menganggap bahwa pembaca mengetahui basis yang digunakan adalah *hexadecimal* contohnya *interrupt 21*, *interrupt 0x13*, dan *interrupt 10h* yang masing-masing ekuivalen dengan *interrupt 0x21*, *interrupt 0x13*, dan *interrupt 0x10*.

Pada tugas besar ini, pemanggilan interrupt dilakukan dengan fungsi berikut.

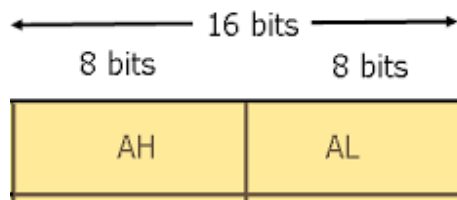


```
interrupt(int number, int AX, int BX, int CX, int DX);
```

**int** number di sini mengacu pada nomor fungsi interrupt yang akan dipanggil misal *interrupt 13h*. Berikut beberapa services yang dapat dilakukan melalui *interrupt 13h*.

Tabel *Interrupt vector 13h (Low Level Disk Services)* [2]

AH	Description
00h	<i>Reset Disk Drives</i>
01h	<i>Check Drive Status</i>
02h	<i>Read Sectors</i>
03h	<i>Write Sectors</i>



Pada OS yang akan dibuat, sebuah integer akan memiliki besar 16 bit. Sebuah integer 16 bit sendiri dapat dibagi menjadi dua bagian 8 bit yaitu **AH** dan **AL**. Untuk pemanggilan *interrupt 13h*, parameter kedua yaitu **int AX** akan terbagi menjadi dua yaitu **AH** dan **AL**. Pada contoh kali ini, dibutuhkan kasus pemanggilan *read sector* yang akan digunakan untuk membaca n-buah *sector*. Pada dokumentasi *INT 13h* (dapat ditemukan pada referensi [2]), untuk memanggil *read sector*, nilai dari **AH** = **0x2**, sedangkan nilai dari **AL** = jumlah sektor yang akan dibaca dalam hexadecimal. Sehingga parameter **AX** = **AH** | **AL** = **AH** + **AL** = **0x0201** yang menunjukkan gabungan dari **AH** = **0x2** dan **AL** = **0x1**. Ingat bahwa 1 digit *hexadecimal* merepresentasikan 4 digit pada *binary*.

Secara umum, penggunaan parameter **AX**, **BX**, **CX**, dan **DX** akan disesuaikan dengan kebutuhan fungsi interrupt yang akan kalian panggil. Jika dibutuhkan kalian dapat melakukan modifikasi pada kode assembly interrupt pada *kernel.asm*.

## 7.3. Referensi Perintah

### 7.2.1 NASM

```
nasm -f as86 <file input> -o <file output>
```

Dalam tugas ini format output yang digunakan adalah as86 (sebuah format *object code* sederhana)

### 7.2.2 ld86

```
ld86 -o <output> -d <object code untuk dilink>
```

Parameter -d menyatakan bahwa hasil *output* tidak memiliki *header* (digunakan pada sistem operasi yang lebih kompleks).

### 7.2.3 BCC

```
bcc -ansi -c -o <output> <input>
```

Parameter -ansi menyatakan versi C yang digunakan (ANSI C) dan -c menyatakan untuk meng-*compile* ke *object code*

## 7.4. Interrupt

- [1] [https://en.wikipedia.org/wiki/BIOS\\_interrupt\\_call](https://en.wikipedia.org/wiki/BIOS_interrupt_call)
- [2] <http://www.oldlinux.org/Linux.old/docs/interrupts/int-html/int-13.htm>
- [3] <http://spike.scu.edu.au/~barry/interrupts.html>

## 7.5. Bacaan Menarik

- [4] [https://yugioh.fandom.com/wiki/Forbidden\\_One](https://yugioh.fandom.com/wiki/Forbidden_One)
- [5] <https://www.quora.com/p/10876/explain-the-interrupt-structure-of-8086-processor-1/>
- [6] [https://wiki.osdev.org/Text\\_mode](https://wiki.osdev.org/Text_mode)
- [7] [https://wiki.osdev.org/Text\\_Mode\\_Cursor](https://wiki.osdev.org/Text_Mode_Cursor)
- [8] [https://wiki.osdev.org/Text\\_UI](https://wiki.osdev.org/Text_UI)
- [9] [https://wiki.osdev.org/Real\\_Mode](https://wiki.osdev.org/Real_Mode)
- [10] <https://users.cs.fiu.edu/~downeyt/cop3402/absolute.html>
- [11] [https://en.wikipedia.org/wiki/Intel\\_8086#Registers\\_and\\_instructions](https://en.wikipedia.org/wiki/Intel_8086#Registers_and_instructions)
- [12] <https://www.includehelp.com/embedded-system/types-of-registers-in-the-8086-microprocessor.aspx>
- [13] [https://wiki.osdev.org/VGA\\_Hardware](https://wiki.osdev.org/VGA_Hardware)
- [14] <https://cs.nyu.edu/courses/fall03/V22.0201-001/combining.html>
- [15] [https://en.wikibooks.org/wiki/X86\\_Assembly/NASM\\_Syntax](https://en.wikibooks.org/wiki/X86_Assembly/NASM_Syntax)
- [16] [https://wiki.osdev.org/Floppy\\_Disk\\_Controller](https://wiki.osdev.org/Floppy_Disk_Controller)
- [17] <https://www.eeguide.com/8086-interrupt/>
- [18] [https://en.wikipedia.org/wiki/BIOS\\_color\\_attributes](https://en.wikipedia.org/wiki/BIOS_color_attributes)
- [19] <http://hilite.me/> (Pretty print untuk source code)
- [20] <https://ascii-tree-generator.com/> (Pembuatan ilustrasi direktori)
- [21] [Repository kating](#)