Tugas Besar
IF2230 - Sistem Operasi
Milestone 01 of ??
“Welcome, Player.”
Pembuatan Sistem Operasi Sederhana
Booting, Kernel, System Call

Dipersiapkan oleh :
Asisten Lab Sistem Terdistribusi

Didukung Oleh :

Waktu Mulai :
Kamis, 11 Februari 2021, 20.21 WIB

Waktu Akhir :
Rabu, 24 Februari 2021, 20.21 WIB

I. Latar Belakang

Sudah satu setengah tahun kamu bekerja sebagai seorang *hunter* rank E dengan susah payah untuk bertahan hidup. Setiap saat kamu keluar dari dungeon, yang kamu dapatkan hanyalah luka dan hinaan sebagai *hunter* paling lemah di dunia. Hingga suatu hari, kamu diajak untuk memasuki *dungeon* rank D. Baru saja kamu memasuki *dungeon* tersebut dan kamu sudah terluka. Dungeon pun berhasil diselesaikan oleh rekan-rekanmu. Mengira kamu akan kembali seperti biasa hanya membawa sebuah serpihan murah, rekanmu yang lain menemukan sebuah pintu masuk lainnya. Demi membiayai adik perempuanmu dan menyembuhkan ibumu, kamu pun tidak ragu untuk masuk ke dalam pintu tersebut.

Setelah membuka pintu tersebut, yang kamu temukan adalah beberapa patung… patung besar yang sangat aneh. Rekan-rekanmu pun mulai mengamati sekeliling. Di antara patung-patung itu, ada sebuah patung yang sangat besar. Seorang rekanmu menyadari bahwa mata patung bergerak dan kemudian… pintu masuk tertutup. Akhirnya seorang rekanmu merasa tidak nyaman dan berusaha untuk keluar menuju pintu… hanya untuk dipisahkan dari barisan. Melihat hal tersebut, semua orang panik, termasuk kamu.

Sang patung besar akhirnya mulai bergerak. Tanpa peringatan apa pun, matanya mengeluarkan laser yang menyerang kamu dan rekan-rekanmu. Ketua kelompokmu pun terpaksa kehilangan tangannya. Kepanikan dan ketakutan semakin menggerogoti tubuhmu dan yang ada hanyalah keputusasaan di hadapan makhluk yang melebihi apa pun. Adrenalin yang

muncul di otakmu pun mendorong kamu untuk berpikir. Kamu pun berhasil menyadari aturan *dungeon*. Namun apakah itu cukup?

Mencoba untuk menggapai patung yang memainkan instrumen, usahamu tidak sia-sia, namun kamu pun menyadari sesuatu. “Kakiku..!”. Kamu yang sudah kehilangan salah satu kakimu, mulai kehilangan harapanmu. Rekanmu berusaha dengan keras untuk memulihkan kakimu, namun dirinya pun sudah kewalahan. Tiba-tiba muncullah sebuah altar. Tidak punya pilihan lain, kalian semua berkumpul di altar. “Pintunya terbuka?!?!”. Tidak hanya itu, seluruh patung mulai mendekati kalian. Kamu pun menyadari bahwa yang perlu kamu dan rekan-rekanmu lakukan adalah menunggu dengan percaya, tetapi.. beberapa rekanmu tidak percaya dengan omong kosongmu dan memilih untuk menyelamatkan diri mereka sendiri. Tidak dapat melihat kemungkinan untuk selamat, kamu pun memilih untuk mengorbankan diri sendiri.

Hingga di saat terakhir pun, kamu memohon.

“Aku tidak ingin mati!”
“Jika saja aku memiliki kesempatan lagi...”

Tiba-tiba, muncul sebuah pesan di otakmu.

“Kamu telah memenuhi semua syarat dari Keberanian Sang Lemah”
“Kamu mendapatkan hak untuk menjadi seorang *player*”

“Apakah kamu akan menerimanya?”
“Ya!”

Akhirnya muncul sebuah pesan yang mengawali perjalananmu menjadi seorang *hunter* yang sebenarnya.

“Halo, *player*.”

II. Deskripsi Tugas

Pada praktikum 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* dengan fungsi:
    - Membaca input dari keyboard
    - Menulis output ke layar
- Mengerti cara kerja interrupt
- Mengerti cara kerja kernel.asm

III. Langkah Pengerjaan

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 18.04/20.04. Pada Ubuntu 18.04/20.04 berikut adalah perintah yang dapat digunakan untuk menginstal program-program di atas**.**

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

3.1. 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.

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

3.2. *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 bootloader.asm -o bootloader
```

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

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

3.3. 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 bagian C dari kernel. Beberapa fungsi sudah disediakan oleh kode kernel.asm untuk digunakan pada file kernel.c. Jika dilihat pada bagian atas kernel.asm terdapat beberapa 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, char character)`
  Fungsi ini menulis sebuah karakter pada *segment* memori dengan *offset* tertentu

- `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 ke *interrupt* tersebut

- `void makeInterrupt21()`
  Fungsi ini mempersiapkan tabel *interrupt vector* untuk memanggil kode Anda jika *interrupt 0x21* terpanggil

- `void handleInterrupt21 (int AX, int BX, int CX, int DX)`
  Fungsi ini dipanggil saat terjadi *interrupt* nomor 0x21. Implementasi *interrupt* 0x21 pada kernel dilakukan di sini

```
int main () {
  putInMemory(0xB000, 0x8000, 'H');
  putInMemory(0xB000, 0x8001, 0xD);
  putInMemory(0xB000, 0x8002, 'a');
  putInMemory(0xB000, 0x8003, 0xD);
  putInMemory(0xB000, 0x8004, 'i');
  putInMemory(0xB000, 0x8005, 0xD);

  while (1);
}

void handleInterrupt21 (int AX, int BX, int CX, int DX) {}
```

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

- 0xB000 merupakan segment video memory
- 0x8000 adalah offset memori yang mendefinisikan karakter pada posisi (x=0, y=0), dimana (x=0, y=0) adalah posisi kiri atas layar.
- 0x8001 adalah offset memori yang mendefinisikan warna dari karakter pada posisi (x=0, y=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 (1) digunakan agar kernel tidak reboot
- void handleInterrupt21 (int AX, int BX, int CX, int DX) harus ada 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 kernel.o kernel.c
nasm -f as86 kernel.asm -o kernel_asm.o
ld86 -o kernel -d kernel.o kernel_asm.o
dd if=kernel of=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

Untuk mempermudah sangat disarankan **membuat script** untuk melakukan kompilasi sistem operasi Anda (mulai dari tahap 1).

Sedikit Teori Penting dan Menarik

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

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

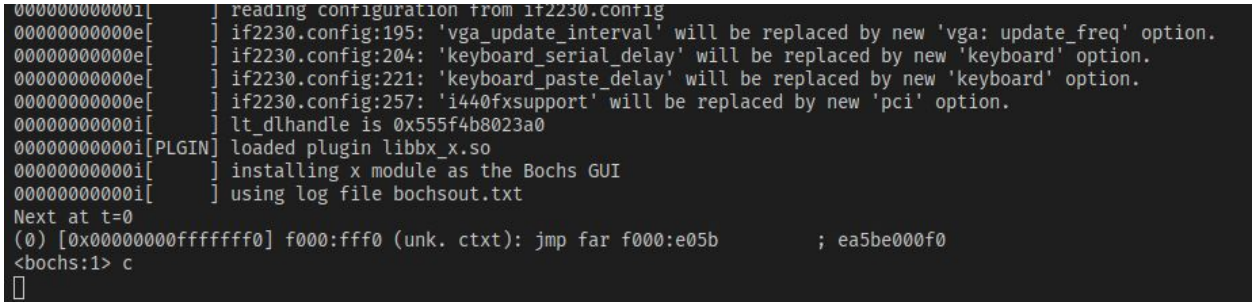
Jika dilihat pada potongan kode di atas, fungsi putInMemory menerima *segment* dan *offset* yang akan dikalkulasikan dengan rumus (segment << 4) + offset. Hal ini diperlukan untuk mengatasi ukuran register yang hanya 16 bit sedangkan alamat memori berukuran 20 bit. Untuk lebih detail bisa melihat referensi pada https://wiki.osdev.org/Real_Mode#Memory_Addressing

3.4. Menjalankan Sistem Operasi

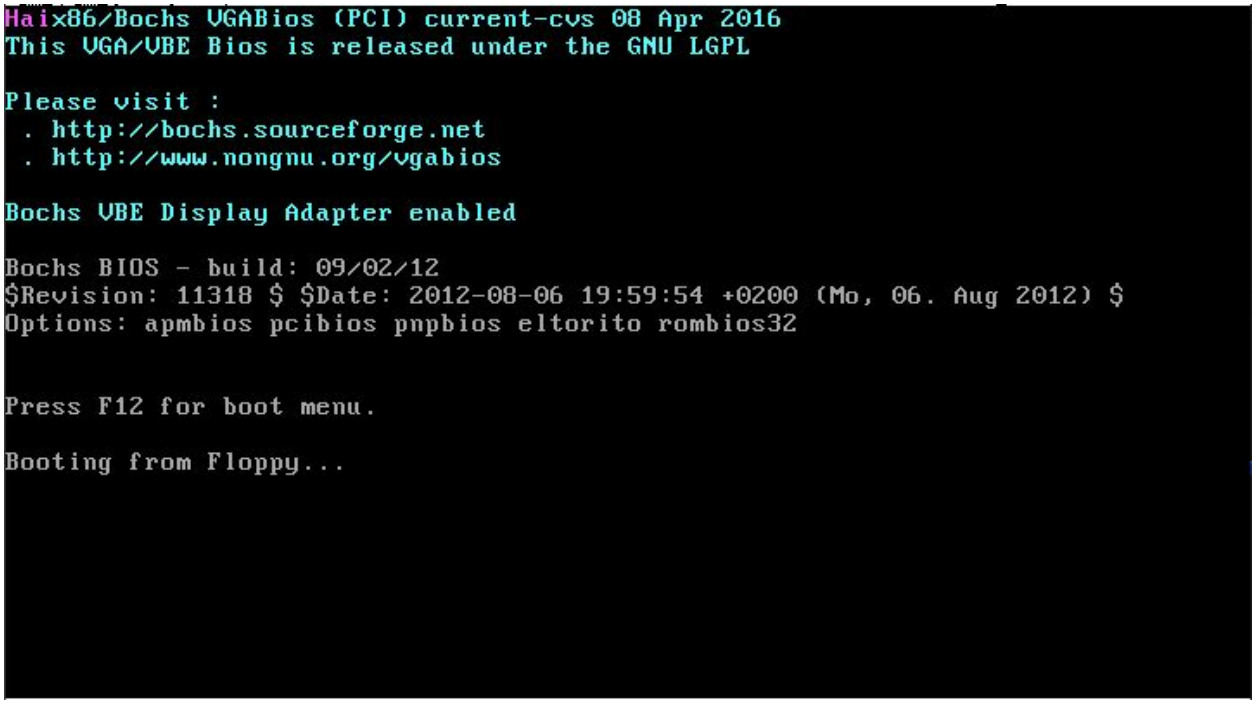
Untuk menjalankan sistem operasi yang Anda buat, akan digunakan emulator bochs. Untuk menjalankannya dibutuhkan sebuah file konfigurasi yang sudah diberikan dengan nama if2230.config

```
bochs -f if2230.config
```

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



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.

```
/* Ini deklarasi fungsi */
```

```
void handleInterrupt21 (int AX, int BX, int CX, int DX);
void printString(char *string);
void readString(char *string);
void clear(char *buffer, int length); //Fungsi untuk mengisi buffer dengan 0

int main() {
  makeInterrupt21();
  while (1);
}

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");
  }
}
```

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 (tulis) dan 0x16 (baca). Untuk penggunaannya dapat melihat referensi di bawah

Sedikit Teori Penting dan Menarik (2) [1]

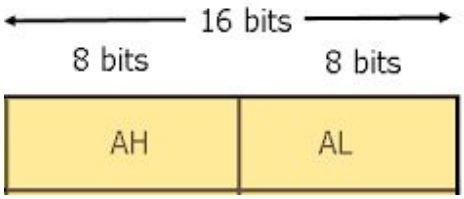
BIOS interrupt calls adalah fasilitas yang diberikan sistem operasi untuk memanggil Basic Input/Output System (BIOS) software pada komputer. BIOS interrupt calls dapat melakukan kontrol langsung pada hardware atau fungsi I/O yang diminta oleh program atau mengembalikan informasi terkait sistem.
Pada arsitektur x86, 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 13. Berikut beberapa services yang dapat dilakukan melalui interrupt 13.

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

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



Pada OS yang kalian buat, 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 13, 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 1 buah sector. Pada dokumentasi int 13 (dapat ditemukan pada reference [2]), untuk memanggil read sector, nilai dari AH = 02h, sedangkan nilai dari AL = jumlah sektor yang akan dibaca dalam hexadecimal. Sehingga parameter AX akan berisi nilai 0x0201 yang menunjukkan gabungan dari AH=02h dan AL=01h. Secara umum, penggunaan parameter AX, BX, CX, dan DX akan disesuaikan dengan kebutuhan fungsi interrupt yang akan kalian panggil.

3.6. Bonus

Bonus untuk milestone ini

- Membuat boot logo dalam ASCII art (nilai +)
- Membuat boot logo dalam mode grafis yang bukan ASCII art (nilai ++)

IV. Pengumpulan dan Deliverables

1. Untuk tugas ini Anda diwajibkan menggunakan *version control system* **git** dengan menggunakan sebuah *repository* ***private*** di Github Classroom "**informatika19**" (gunakan surel *student* agar gratis). Invitation ke dalam Github Classroom "**informatika19"** akan diberikan saat tugas dirilis.

2. Kit untuk pengerjaan dapat diakses melalui link berikut : https://s.id/kit1-os19

3. 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.

4. Lakukanlah *commit* yang wajar (tidak semua kode satu *commit*). Kontribusi Anda di kelompok (lewat commit) akan dinilai.

5. 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* atau *image* hasil kompilasi ke *repository* (manfaatkan **.gitignore**).

6. Mengumpulkan penjabaran cara kerja *interrupt* di sistem operasi Anda dan cara kerja kode di kernel.asm per kelompok dalam bentuk paragraf. Tambahkan penjabaran tersebut dalam bentuk *markdown* (**MILESTONE1.md**) di repository Anda.

7. Isikan nama kelompok dan anggotanya pada link berikut s.id/kelompok-os19, paling lambat tanggal 14 Februari 2021 20.21 WIB.

8. **Mulai** Kamis, 11 Februari 2021, 20.21 WIB waktu server.
   Deadline Rabu, 24 Februari 2021, 20.21 WIB waktu server.
   Setelah lewat waktu *deadline*, perubahan kode akan dikenakan pengurangan nilai.

9. Pengumpulan dilakukan dengan membuat ***release*** dengan tag **v.1.0.0** pada repository yang telah kelompok Anda buat sebelum deadline.

10. Kami akan **menindaklanjuti segala bentuk kecurangan** yang terstruktur, masif, dan/atau sistematis.

11. Diharapkan untuk mengerjakan sendiri terlebih dahulu sebelum mencari sumber inspirasi lain (Google, maupun teman anda yang sudah bisa). Percayalah jika menemukan sendiri jawabannya akan merasa bangga dan senang.

12. Dilarang melakukan kecurangan lain yang merugikan peserta mata kuliah IF2230.

13. Jika ada pertanyaan atau masalah pengerjaan harap segera menggunakan sheets QnA mata kuliah IF2230 Sistem Operasi pada link berikut s.id/qna-prak

14. Sekali lagi, **sangat dianjurkan untuk memakai sistem operasi Ubuntu 18.04/20.04,** karena *Sister Tech Support*™ akan difokuskan pada penggunaan sistem operasi Ubuntu.

V. Tips

1. 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*.

2. Untuk melihat isi dari *disk* bisa digunakan utilitas hexedit.
3. Walaupun kerapihan tidak dinilai langsung, kode yang rapi akan sangat membantu saat *debugging*.
4. Fungsi-fungsi dari *stdc* yang biasa Anda gunakan seperti **mod**, **div**, **strlen**, dan lainnya tidak tersedia di sini. Jika anda mau menggunakannya, anda harus membuatnya sendiri, terutama **mod** dan **div** yang akan sangat berguna.

VI. Referensi Tambahan

6.1. Interrupt

[1] 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

6.2. Referensi Perintah

6.2.1 NASM

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

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

6.2.2 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)

6.2.3 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*

6.3. Bacaan Menarik

[4] http://bit.ly/16BitInterrupts
[5] https://www.ques10.com/p/10876/explain-the-interrupt-structure-of-8086-processo-1/
[6] https://wiki.osdev.org/Text_mode
[7] https://wiki.osdev.org/Text_Mode_Cursor
[8] https://wiki.osdev.org/Text_UI
[9] 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
[12] https://www.includehelp.com/embedded-system/types-of-registers-in-the-8086-microprocessor.aspx
[13] 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
[16] https://wiki.osdev.org/Floppy_Disk_Controller
[17] https://www.eeeguide.com/8086-interrupt/