Jurnal Teknologi Informatika
Vol. 5 No. 2 Juli 2013
Hal. 64 - 84

Membangun *Bootloader* Prosesor x86 dengan Program Debug.com

Tedi Lesmana M, Alexander Waworuntu, Mochamad Rizqian Zulkarnaen
Institut Teknologi dan Bisnis Kalbe, Jakarta

Abstrak:

Bootloader adalah program kecil sebesar 512 bytes yang akan dipanggil pertama kali oleh BIOS dari boot device dan diletakkan di memori utama RAM sebelum sistem operasi diproses. Setelah program bootloader berada di memori dan menjalankan serangkaian proses standar untuk menginisialisasi sistem komputer agar siap memanggil kernel sistem operasi selanjutnya kendali akan diserahkan oleh bootloader kepada kernel sistem operasi. Program bootloader dibangun menggunakan interupsi bawaan yang ada di sistem BIOS dan diterjemahkan oleh instruction set yang ada di prosesor yang hanya bekerja ketika prosesor x86 berada pada mode real. Dalam tulisan ini dijelaskan bagaimana membangun bootloader pada prosesor kompatibel x86 Intel Prosesor menggunakan program debug.com. Pendekatan debug.com untuk membangun bootloader termasuk jarang digunakan karena pada dasarnya program debug.com tidak ditujukan untuk itu. Mengapa dipilih justru karena ternyata sangat efisien dan praktis tidak memerlukan program compiler yang umum digunakan yaitu compiler bahasa assembly, namun demikian tetap dibutuhkan keterampilan penulisan kode-kode program dalam bahasa rakitan prosesor Intel x86. Pemahaman proses booting bootstrap program dari bootloader merupakan hal yang kritikal berkaitan dengan pemahaman sistem yang sangat mendasar untuk pengembangan sistem operasi yang lebih kompleks, dan Intel masih menjaga kompatibilitas teknik booting menggunakan bootloader menggunakan interupsi BIOS dan Instruction Set dari prosesornya pada komputer generasi moderen.Pada gilirannya pemahaman yang mendalam tentang program bootloader dan cara kerja sistem komputer menjalankan program di tahap awal akan sangat bermanfaat untuk pengembangan sistem operasi di tahap selanjutnya.

Keywords : bootloader, interupsi, instruction set, boot device

I. Pendahuluan

1.1. Latar Belakang Penelitian

Salah satu piranti lunak yang sangat pesat perkembangannya adalah sistem operasi. Perusahaan raksasa seperti Microsoft dan Apple sejak awal sangat intensif dan memiliki fokus yang sangat kuat pada penelitian dan pengembangan piranti lunak sistem operasi. Sistem operasi yang sangat penting di komputer-komputer besar seperti *mainframe* dan mini komputer tidak terlalu banyak ditemui oleh pengguna sistem komputer, tetapi ketika ukuran komputer menjadi mikro dan pada saat ini sudah menjadi komputer genggam, sistem operasi terasa demikian dekat dalam kehidupan masyarakat.

Kepopuleran sistem operasi di masa komputer personal (PC) di era 80-an sampai dengan era 90-an sudah makin tergeser dengan kepopuleran sistem operasi di era 2000-an hingga

abad 21. Semakin bervariasi dan semakin lengkap dengan berbagai macam fitur unggulan dan kemudahan pengoperasian. Selain semakin kompleks dan lengkap, berbagai macam *platform hardware* juga memicu perkembangan sistem operasi di berbagai macam prosesor yang tidak lagi didominasi oleh Intel x86. Apalagi di jaman sekarang ketika teknologi telepon genggam demikian kuat didukung oleh spesifikasi piranti keras yang terkini dan tidak lagi didominasi oleh satu dua produk. Hal ini diikuti dengan persaingan pengembangan sistem operasi dan aplikasi di atasnya yang juga semakin sengit.

Di Indonesia, kebanyakan masyarakat masih berada dalam tahap pengguna yang sibuk dengan berbagai macam produk yang disediakan dari luar. Belum banyak penelitian mendalam untuk membangun sistem operasi di berbagai macam *platform* mulai dari PC sampai dengan telepon pintar / *smartphone*. Begitu panjang proses penelitian yang harus dilakukan karena menuntut pengetahuan dan pemahaman kerja sistem komputer yang rumit. Penelitian ini mencoba untuk menjadi awal dari penelitian yang panjang tersebut. Tujuan dari penelitian ini untuk memberikan pemahaman dasar dan pokok dari awal kerja sebuah sistem operasi. Tulisan ini memberikan pemahaman yang sangat mendasar dan penting bagaimana sebuah sistem operasi bekerja di atas sistem komputer, karena pada dasarnya sistem operasi adalah piranti lunak dan harus bekerja di atas piranti keras yaitu sistem komputer.

Peralihan kendali dari piranti keras ke piranti lunak tidak dapat langsung dikerjakan oleh sistem operasi. Sebuah sistem operasi membutuhkan sebuah program kecil yang menjadi antara proses peralihan tersebut. Program ini dinamakan *bootloader*. Pada dasarnya sistem komputer yang terdiri dari komponen dasar prosesor, memori, I/O *device* dan sistem bus telah diatur untuk memanggil atau memproses program *bootstrap* yang berada di sebuah *boot device*. *Bootstrap* ini dinamakan *bootloader*. *Bootloader* memiliki tanggungjawab utama yaitu menyerahkan kendali dari piranti keras sistem komputer kepada sistem operasi. Apabila tugas tersebut telah selesai dilakukan maka kendali sistem komputer selanjutnya dipegang oleh sistem operasi. Meskipun sangat singkat prosesnya, tanpa bantuan *bootloader* sistem operasi tidak mungkin dapat bekerja, karena program inti sistem operasi yaitu kernel terlalu besar dan sistem BIOS tidak dapat memanggil lebih dari sebuah *sector* penyimpanan pada *boot device*. Di sinilah peran penting *bootloader* yang meskipun kecil tetapi berfungsi secara kritis.

Untuk pengembangan program *bootloader* sebelumnya harus dipahami cara kerja sistem komputer secara mendalam, baik sistem piranti keras dan bagaimana sistem piranti lunak bekerja di atas sistem komputer. Untuk itulah

pembuatan sistem operasi tidak bisa lepas daripada pemahaman sistem peranti keras dan sistem pendukungnya seperti BIOS dan *bootloader*.

1.2. Rumusan Masalah

Dalam penelitian ini dirumuskan masalah pokok yang menjadi benang merah penelitian yaitu bagaimana membangun program *bootloader* pada prosesor Intel x86 menggunakan program debug.com.

1.3. Tujuan Penelitian

Dalam tulisan ini akan disasar dua buah tujuan yang menjadi pijakan penting pengembangan program *bootloader* yaitu

1. Membuat program *bootloader* yang bekerja pada lingkungan prosesor Intel x86.
2. Membuat sebuah keluaran sebagai pengganti pemanggilan kernel sistem operasi.

II. Kajian Pustaka

2.1. Sistem Komputer

Sistem komputer terdiri atas tiga komponen utama yaitu mikroprosesor, memori utama yang biasanya berupa RAM (*Random Access Memori*) dan komponen I/O (*Input/Output*). Ketiga komponen ini dihubungkan melalui sistem bus. Sistem bus adalah jalur-jalur penghubung ketiga komponen yang terdapat pada papan utama rangkaian elektronik. Jalur-jalur penghubung ini biasanya berupa jalur-jalur tembaga.

Dalam setiap pekerjaan yang terjadi di sistem komputer, mikroprosesor atau prosesor menjadi aktor utama yang memulai, menggerakkan dan mengendalikan keseluruhan proses kegiatan. Program-program yang terdapat pada komponen input / output akan diproses di memori melalui kendali prosesor. Memori pada dasarnya digunakan sebagai penampungan sementara sebelum setiap *byte* yang ada di tiap-tiap lokasi memori akan di-*fetching* melalui jalur-jalur bus ke dalam register prosesor (CPU). Ada tiga jalur bus yang menghubungkan tiga komponen besar prosesor, memori dan I/O yaitu *address bus*, *data bus*, dan *control bus*. Masing-masing bus memiliki peranan masing-masing berkaitan dengan komponen-komponen dan proses yang menggunakannya [1].

Di dalam prosesor terdapat tiga komponen pokok juga yaitu register itu sendiri sebagai tempat penampungan akhir *byte-byte* yang akan diproses, *Arithmethic Logical Unit*, dan *Control Unit*. Tiga komponen pokok dan kunci di dalam prosesor ini yang saling bekerjasama menerjemahkann setiap instruksi yang masuk dari RAM untuk diproses dan diterjemahkan ke dalam tindakan pengendalian sistem komputer, hasilnya akan diberikan kembali ke proses yang berada di dalam RAM atau diberikan kepada komponen I/O [1].

2.2. Sistem Operasi

Sistem operasi adalah program yang dirancang untuk melakukan fungsi pokok sistem operasi yaitu *process management*, *process communication management*, *memory management*, *file management*, *I/O management*, *security and protection management*, dan *network management* [2].

Ketujuh fungsi pokok ini menjadi tugas utama program sistem operasi. Dalam pengelompokkannya program-program yang menjalankan fungsi-fungsi tersebut dinamakan kernel sistem operasi, sehingga sistem operasi dalam pengertian fungsi hanya dikatakan sebagai sebuah kernel. Kernel sistem operasi memiliki tugas melakukan tujuh fungsi sistem operasi yang didukung oleh program-program lain sebagai pembantunya seperti *shell*, *compiler* dan *user interface*.

2.3. *Booting*

Booting adalah proses pengenalan media boot oleh sistem BIOS yang dijadikan media penyimpanan program inisiasi ketika pertama kali sistem komputer diaktifkan. Media boot ini biasanya berupa disk seperti *floppy disk*, *harddisk*, CD/DVD ROM, atau *flash disk*.

Sistem komputer tanpa sistem operasi dikendalikan oleh sistem BIOS yang difungsikan untuk memeriksa kelengkapan dari sistem komputer pada proses POST (*Power OnSelf Test*). Proses POST menjadi langkah pemeriksaan sistem komputer oleh BIOS sebelum BIOS menyerahkan kendali sistem komputer terhadap sistem operasi [5].

Sistem BIOS sudah dibuat oleh produsen sistem komputer sehingga bersifat *pre-build*. Sistem BIOS diletakkan di dalam sebuah chip PROM (*Programmable Read Only Memory*) atau EEPROM (*Erasable Electrical Programmable Read Only Memory*). Sistem ini hanya terdiri dari program kecil dan pendek, dan tidak mungkin berisi program kompleks yang dapat memenuhi seluruh kebutuhan pengguna. Untuk itu sistem BIOS akan menyerahkan kendali kepada sistem operasi. Langkah penyerahan kendali ini dilakukan dengan memanggil media *boot* yang biasanya dalam bentuk *disk*, media tersebut kemudian disebut sebagai *boot device*.

Tahapan penting yang dilakukan dalam proses *booting* adalah sistem BIOS akan meletakkan program yang terdapat di media *boot* ini pada memori RAM di lokasi memori 7C00, sedangkan program tersebut biasanya disebut *bootloader* harus berada di dalam media yang akan dikenali oleh sistem BIOS.

Jika media boot ditemukan, sistem BIOS akan mencari program bootloader di lokasi *boot sector*. Apabila media *boot* tidak ditemukan sistem BIOS akan menampilkan pesan kesalahan. Jika sistem BIOS menemukan program *boot* maka kendali akan berpindah ke program *bootloader*, begitu program tersebut

sukses ditempatkan di lokasi memori 7C00, karena selanjutnya instruksi yang akan diproses dimulai pada lokasi memori 7C00 tersebut. Pada titik inilah terjadi peralihan penting antara kendali sistem komputer oleh BIOS berpindah kepada kendali sistem komputer oleh program lain buatan pengguna [5].

2.4. *Bootstrap*

Bootstrap adalah program yang terdapat pada *bootloader*. *Bootstrap* menempati ruang sebanyak satu *sector* pada media penyimpanan (*storage* media). Media penyimpanan berupa disk memiliki struktur penyimpanan yang terdiri dari *track*, *sector* dan *head*. *Boot sector* adalah lokasi pada media penyimpanan yang digunakan sebagai *boot disk* yang berisi program *bootstrap*. *Boostrap* menempati ruang ini sebesar 512 bytes, di mana 510 bytes berisi program dan 2 bytes berisi identitas sector sebagai *boot* disk yaitu angka 55AA heksadesimal. Ketika *bootstrap* berada di *sector*, *bootstrap* tidak mengandung alamat, tetapi ketika diletakkan di memori oleh sistem BIOS *bootstrap* harus diletakkan di offset 7C00 agar dapat dikenali oleh prosesor dan instruksi di lokasi tersebut langsung diproses melalui petunjuk IP (*Instruction Pointer*).

2.5. Debug.com

Debug atau debug.com adalah program 16 bit yang telah ada pada sistem operasi DOS buatan Microsoft. Program ini masih terus dipertahankan kehadirannya hingga pada sistem operasi Windows XP dan Windows 7 versi 32 bit.

Program debug akan menciptakan ruang memori di memori RAM (di luar area yang sudah digunakan oleh sistem operasi, aplikasi dan program lainnya) untuk dapat digunakan membuat program menggunakan instruksi-instruksi assembly / bahasa assembly [3].

MEMORI	
ADDRESS	ISI MEMORI
FFFFF	23
FFFFE	
FFFFD	
FFFFC	AA
FFFFB	
FFFFA	BC
FFFF9	
FFFF8	DD
00002	
00001	55
00000	

Gambar 1. Ilustrasi Struktur Memori RAM

Seperti dapat dilihat pada Gambar 1., Struktur dari memori RAM terdiri atas bagian alamat / *address*, dan bagian isi dari alamat tersebut.

Setiap program, aplikasi, atau *software* yang akan dieksekusi oleh prosesor, harus berada di memori (RAM). Oleh karena itu memori RAM penuh dengan berbagai macam program yang

akan atau sedang dieksekusi oleh prosesor.

Sebagai contoh, jika membuat program yang menggunakan compiler, kemudian keluaran program akan muncul dalam bentuk program .exe atau program *scripting* yang dieksekusi oleh *browser* atau *web server*. Pada akhirnya semua program harus diletakkan di alamat memori untuk dapat dieksekusi oleh prosesor.

Begitu juga dengan program debug yang sudah dieksekusi, lokasinya akan berada di memori. Tetapi program debug memiliki keunikan, karena selain program ini menempati lokasi memori, debug.com juga menyediakan ruang di memori di mana dapat ditulisi baris-baris instruksi di sana. Kemudian setiap *programmer* untuk menulis baris-baris instruksi langsung di memori tersebut maka harus menuliskannya dalam bentuk bahasa *assembly* dengan kode-kode program yang dinamakan *mnemonic*. *Mnemonic* adalah instruksi-instruksi yang akan memerintahkan prosesor untuk menampung dan mengolah nilai-nilai yang diberikan kepada *register-register*. Semua program yang ada di memori pada akhirnya hanya akan diproses oleh prosesor melalui penampung terkecil dalam sistem komputer yaitu register, yang lokasinya ada di dalam prosesor. Informasi, data atau nilai yang ada di baris-baris kode program di memori akan disampaikan kepada *register* yang ada di dalam prosesor melalui jalur-jalur penghantar yang dinamakan data bus. Beginilah konsep prosesor mengeksekusi program dalam sistem komputer.

Bagaimana debug berada di memori dan bagaimana debug menyediakan ruangan di memori untuk membuat program dapat diilustrasikan seperti pada Gambar 2.

MEMORI	
ADDRESS	**ISI MEMORI**
FFFFF	Sistem Operasi
FFFFE	Sistem Operasi
FFFFD	Sistem Operasi
FFFFC	debug
FFFFB	debug
FFFFA	debug
FFFF9	mem alocation
FFFF8	mem alocation
FFFF7	mem alocation
00002	
00001	55
00000	

Gambar 2. Alokasi Memori Untuk Program

Seperti terlihat pada gambar bahwa, sistem operasi dan debug menempati area memori tertentu, sedangkan ada juga area memori yang sudah dipesan untuk tempat program-program yang dibuat dengan menggunakan debug. Pengaturan ini sendiri dilakukan oleh sistem operasi, di mana program debug.com berjalan.

2.6. Sistem Interupsi

Sistem interupsi adalah sistem *library* perintah-perintah *hardware* yang

terdapat pada chip BIOS. Sistem interupsi akan memudahkan programmer untuk melakukan pengendalian sistem *hardware* di awal inisiasi sistem ketika sistem operasi belum dapat bekerja. Sistem interupsi hanya dapat bekerja pada mode *real* dari prosesor x86. Untuk itu penggunaan sistem interupsi sangat berkaitan erat dengan teknik pemrograman pada mode *real* prosesor. Di lain pihak prosesor dapat juga dioperasikan pada mode *protected* yang dalam tulisan ini berada luar bahasan.

Pada mode *real*, *programmer* dapat mengakses library sistem interupsi dari BIOS yang diletakkan di bagian bawah memori RAM. Pada mode real jumlah memori RAM yang dapat diakses oleh program sebesar 1 Mbytes alamat. Oleh karena itu pada mode *real* jumlah memori yang dapat diakses sangat terbatas. Kelebihannya adalah tersedia cukup banyak fungsi kendali sistem komputer yang disediakan oleh sistem interupsi. Beberapa *library* fungsi atau sistem interupsi yang disediakan oleh interupsi BIOS seperti pengolahan teks, pengolahan grafik, proses baca dan tulis ke media, akses sistem waktu, dan prosesor [3].

2.7. *Instruction Set*

Instruction set berada di dalam prosesor dan bersifat *firmware*, artinya sudah tertanam dalam rancangan prosesor. *Firmware instruction set* memungkinkan prosesor memahami apa yang harus dilakukan ketika register berisi instrusi-instruksi *mnemonic* yang sudah diterjemahkan ke dalam data biner. Jadi *instruction set* berfungsi sebagai referensi utama yang paling dasar bagaimana prosesor memahami apa yang harus dilakukan oleh. Untuk itu *instruction set* sangat tergantung pada prosesor. Di sinilah *instruction set* x86 atau prosesor-prosesor Intel memiliki kesamaan. *Instruction set* untuk prosesor produksi pabrik lain dapat saja kompatibel apabila menjaga kompatibilitasnya dengan seri prosesor x86 atau sama sekali berbeda. Jika *instruction set* pada prosesor berbeda, maka jenis, jumlah, dan fungsi dari *mnemonic-mnemonic* yang ada akan memiliki perbedaan. Meskipun banyak yang memiliki fungsi yang sama, tapi seringkali memiliki sintaks penulisan yang berbeda [5].

2.8. *Real Programming*

Mode pemrograman *real* adalah mode pemrograman yang hanya dapat mengakses memori RAM tidak lebih besar dari 1 Megabytes. Pada dasarnya mode pemrograman *real* merupakan mode yang standar akan diaktifkan ketika prosesor melakukan inisiasi. Produsen mempertahankan mode ini untuk menjaga kompatibilitas program-program lama yang sudah banyak diproduksi menggunakan mode *real*. Mode *real* tidak digunakan untuk mengakses memori dengan kapasitas besar pada komputer modern karena

keterbatasannya. Mode *real* memiliki keunggulan karena kesederhanaan pengoperasiannya dan pemrogramannya, dan juga karena didukung oleh *library* standar untuk mengakses sistem. Mode *real* menggunakan sistem pengalaman SEGMENT:OFFSET yang masing-masing terdiri dari 16 bit menggunakan register 16 bit. Dengan kombinasi dua buah register 16 bit, akan diperoleh pengalamatan absolut 20 bit yang dinyatakan dalam notasi heksadesimal 5 digit heksadesimal. Register-register yang umum digunakan pada model real seperti AX, BX, CX, DX dan CS, DS, ES, SS serta IP, BP, SP juga register SI dan DI [5].

III. Metodologi Penelitian

3.1. Pendekatan Penelitian

Pembuatan program *bootloader* dilakukan pada sebuah *boot device*. *Boot device* adalah piranti yang berisi program yang pertama kali akan diproses oleh sistem BIOS (Basis Input Output System). *Boot device* yang umum digunakan adalah *floppy disk*, *harddisk*, CD/DVD ROM, *network adapter*, dan chip boot ROM. *Boot device* yang akan dipilih sebagai percobaan adalah *floppy disk*. Pemilihan dilakukan karena struktur dari *floppy disk* ini sederhana, mudah digunakan tidak membutuhkan ruang yang besar dan memiliki kesamaan struktur untuk *boot device* populer lainnya seperti *harddisk*, CD/DVD ROM dan USB drive.

Selanjutnya akan dibangun lingkungan *virtual*, yaitu komputer *host* sebagai *real* komputer dan komputer *virtual* yaitu *guest* dibangun di atas komputer *real* menggunakan program virtualisasi VMWare Player. Komputer *real* digunakan untuk membangun program *bootloader*, sedangkan komputer virtual digunakan untuk melakukan ujicoba program. *Virtual floppy disk*-nya akan digunakan sebagai media *boot device*.

3.2. Unit Analisis Penelitian

Bagian pokok dari *bootloader* yang akan dianalisis adalah *bootstrap*. *Bootstrap* adalah program utama yang menjadi bagian dari bootloader. Di sini program harus dirancang untuk ditempatkan pada lokasi memori tertentu di RAM akan dikenal oleh sistem BIOS dan selanjutnya dapat dieksekusi. *Bootstrap* kemudian akan menjalankan sebuah eksekusi keluaran sebagai tanda bahwa *bootloader* berhasil dieksekusi.

3.3. Metode Pengumpulan Data

3.3.1. Penelitian Pendahuluan

Penulis sebelumnya melakukan penelitian pendahuluan terhadap beberapa macam *bootloader* yang sudah ada. Penelitian pendahuluan bertujuan untuk melakukan pemetaan masalah dan batasan kasus serta pemilihan teknologi pengembangan program *bootloader*. Pada penelitian pendahuluan dipilih *platform bootloader* khusus untuk prosesor Intel

x86 (segala macam jenis prosesor yang kompatibel dengan prosesor Intel). Kemudian dipelajari juga bagaimana sistem komputer bekerja dan cara kerja beberapa program *bootloader* yang sudah jadi. Pada penelitian ini penulis juga melakukan ujicoba *bootloader* pada lingkungan virtual. Setelah dilakukan penelitian pendahuluan penulis memutuskan untuk menggunakan cara pengembangan program *bootloader* tanpa menggunakan *compiler* bahasa assembly. Ada cara praktis yang lebih mudah untuk pembuatan program *bootloader* yaitu dengan menulis kode-kode program *assembly* langsung di memori RAM untuk kemudian disalin ke *boot device* yang semuanya dapat dilakukan dengan menggunakan program debug.com yang disediakan oleh DOS keluaran Microsoft atau Windows XP, dan Windows 7 versi 32 bit. Jika akan menggunakan versi tidak berbayar maka dapat menggunakan Free DOS.

3.3.2. Studi Kepustakaan

Ada beberapa studi kepustakaan penting yang mendukung penelitian seperti bahasan tentang dasar-dasar sistem operasi, penjelasan tentang sistem komputer, proses *booting*, sistem BIOS, sistem interupsi, arsitektur prosesor, *instruction set*, dan teknik pemrograman mode *real*. Bahan lain yang dijadikan sumber bacaan dan referensi adalah pustaka *online* dari para pengembang yang telah membuat beberapa macam contoh program *bootloader* dengan beberapa model dan bahasa pemrograman *assembly*.

3.4. Metode Pengembangan Program

Proses pengembangan program melewati tahapan : perancangan program - pengkodean - uji coba - dan debug / perbaikan. Program *bootloader* adalah program yang sangat kecil, sehingga tidak dibutuhkan sistem pengembangan yang kompleks seperti menggunakan tahapan utuh dari *Software Development Life Cycle*. Tahapan yang digunakan di sini dirasa cukup untuk keperluan praktis pembuatan program *bootloader* berdasar *best practice* teknik pemrograman sehari-hari.

3.4.1. Analisis Masalah Penelitian

Program *bootloader* adalah program sebesar 512 bytes yang harus diletakkan di *boot device* di bagian *sector* awal dari partisi awal *boot device* tersebut. *Boot device* adalah eksternal device yang merupakan komponen *Input*/*Output* sistem komputer. Wujud dari *boot device* ini biasanya berturut-turut sesuai evolusinya seperti *floppy disk*, *harddisk*, CD ROM, DVR ROM, dan *flash drive*.

Dari berbagai macam jenis *boot device*, sistem BIOS akan mengenal *boot device* secara hampir seragam yaitu dalam struktur penyimpanan yang terdiri atas *track* dan silinder, *sector*, *head*, dan partisi. *Bootloader* adalah program yang

dinamakan *bootstrap* yang berada di lokasi partisi aktif, sector 0, *track* / silinder 0, dan *head* 0 dengan kode *bootloader* sebesar dua bytes yaitu angkat heksadesimal 55AA.

Sistem komputer adalah piranti keras. Bagian yang mengandung unsur-unsur piranti lunak yaitu berisi kode-kode program yang sudah tertanam (*embedded*) pada saat pabrikasi adalah *instruction set* yang ada di dalam prosesor dan sistem interupsi yang terdapat pada BIOS.

Semua kode piranti lunak paling dasar yang sudah tertanam pada komponen prosesor dan BIOS tersebut pada dasarnya tidak mampu digunakan untuk mengelola sistem komputer dan sistem piranti lunak yang lebih kompleks dan kapasitasnya juga terlalu kecil, sehingga dibutuhkan program yang lebih kompleks dan juga lebih besar ukurannya. Program seperti ini dinamakan *kernel* sistem operasi. Kernel sistem operasi meskipun masih dapat dikatakan kecil tetapi sudah memiliki kemampuan dasar sistem operasi yang dinyatakan dalam standar POSIX (*Portable Operating System Interface*). Namun demikian ukurannya masih terlalu besar jika harus ditanam dalam *chip* / komponen dasar seperti prosesor dan BIOS. Oleh karena itu kernel sistem operasi dan program lainnya biasanya disimpan di dalam *secondary storage* seperti *harddisk*.

Masalahnya adalah kernel sistem operasi yang masih berada di dalam *secondary storage* tidak dapat diproses oleh prosesor jika belum dimasukkan ke dalam memori utama seperti RAM. Untuk itu *bootloader* berperan sangat penting sebagai penanggungjawab yang mengalihkan kernel untuk diletakkan di dalam memori utama RAM untuk selanjutnya diproses oleh prosesor.

Sejak BIOS pertama kali dipicu pada saat komputer dinyalakan, BIOS tidak perlu memanggil langsung *kernel* sistem operasi karena ukurannya yang sangat besar, tapi cukup memanggil program bantu yaitu *bootloader* yang ukurannya kecil, untuk kemudian memanggil sistem operasi. Pada gilirannya *bootloader* sendiri hampir selalu menjadi bagian dari sistem operasi karena kedekatan fungsinya ini.

Pada tulisan ini, *bootloader* yang dibangun tidak digunakan untuk memanggil sistem operasi, tetapi hanya digunakan untuk menampilkan sebuah keluaran sederhana. Tujuannya untuk memahami cara kerja sistem komputer dan proses sistem operasi bekerja untuk pengembangan lebih lanjut. Sistem Operasi adalah program yang kompleks, untuk memahaminya perlu dilakukan dengan cara tahap demi tahap, untuk itulah penelitian yang dilakukan pada tulisan ini membatasinya pada bagian *bootloader* terlebih dahulu.

3.4.2. Proses Pembangunan Program

Tahap-tahap pembangunan program akan meliputi empat proses besar

yaitu perancangan program, pengkodean, uji coba, dan debug. Perancangan program akan meliputi analisis terhadap kebutuhan sistem (*user requirements* dan *system requirements*), tahap pengkodean meliputi proses pembuatan kode program dan bagaimana kode-kode tersebut dibangun, tahap uji coba adalah tahap bagaimana kode-kode program tersebut diujicoba pada lingkungan virtual, dan tahap debug dilakukan untuk memperbaiki program.

3.4.2.1. Perancangan Program

Bootloader adalah program yang ditulis menggunakan bahasa assembly, menggunakan sintax assembly sesuai dengan *compiler* yang digunakan. Cara lain adalah menggunakan mnemonic yang dimengerti oleh instruction set prosesor Intel x86. Untuk itu perlu dipersiapkan spesifikasi program yang akan dibuat seperti pada tabel 1:

Tabel 1 Spesifikasi Program Bootloader

KRITERIA	KAPASITAS
Ukuran program	512 bytes
Starting Address	7C00h di RAM
Posisi disk	*Boot sector*
Bahasa	*Assembly* / Sintaks *mnemonic*
Platform	Prosesor x86 (IBM Kompatibel)
Tahap eksekusi	1.*Power on* 2.Prosesor 3.BIOS 4.POST 5.*Boot device* 6.*Bootstrap loading* 7.Output / keluaran
Media pengembangan	*Real computer*, *virtual computer*, *floppy disk*
Software tools	VMWare Player, Imdisk 32/64 bit, Free DOS, debug.com, Windows 7 32 bit/Windows XP, text editor

3.4.2.2. Pengkodean

Pembuatan program dilakukan di teks editor, kemudian kode program langsung ditulis ke lokasi *boot sector* dengan menggunakan perintah yang ada di program debug.com. Di sini penulis menggunakan teknik langsung tidak seperti kebanyakan teknik yang digunakan yaitu menggunakan *compiler assembly language* untuk melakukan kompilasi program dan menuliskannya di lokasi sector yang diinginkan.

Untuk itu tahapannya dapat dirunut sebagai berikut :

1. Pembuatan kode program
2. Ditulis di teks editor, termasuk memasukkan instruksi *write* ke *boot sector* pada debug
3. Ditransfer ke debug menggunakan perintah *pipeline*

Atau dapat juga menggunakan program debug.com langsung sebagai berikut :

1. Masuk ke debug.com
2. Menulis program
3. Melakukan transfer ke *boot sector* menggunakan *command debug.com*

Program yang ditulis, pada dasarnya adalah baris instruksi yang dibaca oleh instruksi BIOS untuk ditempatkan di alamat memori 7C00 dan baris instruksi ini

selesai. Untuk itu agar proses ini terlihat maka diperlukan program tampilan untuk menjadi tanda bahwa memang program bootloader berhasil dikenal oleh sistem BIOS.

3.4.2.3. Uji Coba

Setelah kode-kode program telah ditransfer ke *bootsector* dari *image floppy disk*. Maka *image floppy disk* akan dianggap sebagai *boot device*. Selanjutnya adalah melepaskan (*unmount*) *image floppy disk* dari komputer host dan memasangnya (*mounting*) ke komputer *guest*. Selanjutnya komputer *guest* diaktifkan / dinyalakan dan BIOS akan membaca *boot sector* dari *image floppy disk* dan mengeksekusi *bootstrap* / program *bootloader* dan menghasilkan tampilan yang direncanakan sebagai tanda bahwa program bootloader telah berhasil diletakkan di lokasi memori 7C00 heksadesimal.

3.4.2.4. *Debug*

Proses pelacakan kesalahan dan perbaikan dilakukan jika hasil yang diharapkan tidak tercapai, maka *image floppy disk* harus di-*unmount* kembali dan kembali dipasang pada komputer *host*. Selanjutnya proses koding dilakukan kembali dan mengulang proses transfer kode-kode *bootloader* tersebut ke *bootsector*. Tahapan ini terus dilakukan sampai hasil yang diharapkan sesuai dengan yang akan dicapai oleh kebutuhan perancangan di awal pengembangan program *bootloader*.

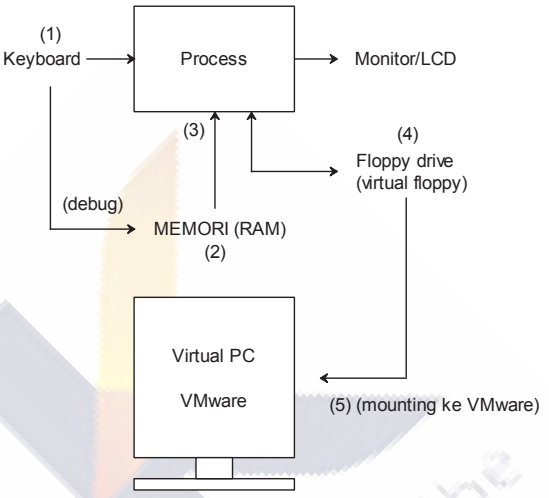
IV. Analisis dan Pembahasan

4.1. Rancangan Program

Setelah sebelumnya sudah direncakan spesifikasi dari program *bootloader* seperti pada Tabel 1, maka ukuran program sudah direncanakan sebesar 512 bytes. Ukuran 512 adalah ukuran standar sebuah *sector* pada media penyimpanan *floppy disk*. Karena di dalam percobaan ini digunakan *image floppy disk* yang akan dipasang pada teknologi *virtual* VMWare *player* maka ukuran *sector* pada *disk floppy image* mengikuti ketentuan yaitu sebesar 512 bytes per *sector*. Ukuran ini tidak semuanya digunakan untuk program, karena dua *bytes* terakhir dari *sector* akan berisi data pengenal yaitu 55AA. Angka heksadesimal ini merupakan ketentuan umum dan standar yang akan dijadikan pengenal oleh BIOS bahwa *sector* tersebut akan dianggap sebagai *boot sector* atau *sector* yang akan pertama kali dipanggil *bootstrap*-nya dan diletakkan di memori oleh program BIOS. Lokasi alamat di memori RAM yaitu di alamat 7C00 heksadesimal. Lokasi 7C00 heksadesimal ini merupakan alamat OFFSET. Karena program di sini terlalu kecil dengan kapasitas yang terbatas, maka bahasa pemrograman yang paling efisien digunakan adalah bahasa rakitan atau bahasa *assembly*. Teknik pemrogramannya menggunakan program debug.com yang

dapat mengakses langsung ke memori RAM dan program langsung diketikkan di sana dan lalu di-transfer ke lokasi yang dituju yaitu *boot sector* dari *image floppy disk*. Jadi di sini tidak digunakan *compiler assembly* seperti TASM, NASM, GASM atau MASM. Selanjutnya program akan ditempatkan di media *floppy disk*. Karena media ini sekarang sudah sulit didapatkan secara fisik, maka digunakan *image floppy disk* yang dapat dikenali oleh program virtualisasi VMWare *Player* dan dianggap sebagai eksternal *drive floppy drive* yang dapat digunakan untuk melakukan proses *booting*. Dengan menggunakan media *virtual* juga akan melindungi lingkungan pengembangan dari kerusakan yang fatal, karena percobaan akan melakukan proses penulisan dan pembacaan ke sector yang vital yaitu *boot sector*, sehingga apabila terjadi kesalahan tidak akan merusak sistem produksi atau lingkungan pengembangan. Semua proses percobaan akan dilakukan di lingkungan virtual. Lingkungan dari proses pengembangan dapat dilihat secara rinci pada Gambar 3.

Gambar 3. adalah lingkungan yang dibangun untuk mengembangan program *bootloader*. Terdapat komputer nyata dengan sistem operasi Windows XP atau Windows 7 32-bit di mana pada mode *Command Prompt* memiliki program debug.com. Ketika kita memanggil program debug.com, program ini akan mengambil rentang memori tertentu di memori RAM sistem Windows XP atau



Gambar 3. Lingkungan Pengembangan Bootloader

Windows 7 untuk digunakan oleh aplikasi debug.com. Jadi pada dasarnya area memori ini sudah eksklusif digunakan oleh program debug.com. Program debug.com termasuk menyediakan area memori tertentu tempat penulisan kode-kode program nantinya. Jadi kode-kode program yang sudah ditulis dan nanti akan dieksekusi pada dasarnya berada di ruang memori aplikasi debug.com, namun demikian program-program tersebut tetap memiliki akses langsung ke sistem *hardware* termasuk berbagai macam sistem interupsi BIOS.

Program-program atau *mnemonic* dan instruksi *assembly* yang ditulis di lingkungan debug.com akan berada langsung di tiap alamat memori RAM, sehingga kode-kode instruksi tersebut dapat langsung dieksekusi tanpa perlu melakukan kompilasi lagi. Hal ini dapat dipahami karena kode instruksi *mnemonic* oleh debug.com dapat langsung

diterjemahkan ke dalam bentuk biner, dan langsung dimengerti oleh prosesor x86 dalam library *Instruction Set*-nya. Untuk itulah mengapa dipilih *platform* prosesor x86, karena kode *bootloader* ini hanya seragam di lingkungan *instruction set* yang sama yaitu prosesor-prosesor yang memiliki kompatibilitas dengan keluarga prosesor Intel Corp.

Setelah program selesai dibuat maka, kode program dapat langsung ditransfer untuk ditempatkan di *boot sector* dari *image floppy disk* yaitu tepatnya di *sector* 0, *head* 0, *track* 0, *drive* 0 atau *floppy drive*. Sebesar 512 bytes data ditransfer ke image floppy drive dengan perintah yang disediakan di program debug.com. *Image floppy drive* yang sudah diberi *bootstrap bootloader* kemudian akan dipasang (*mount*) pada komputer virtual VMWare *player*. Tidak ada bentuk sistem operasi khusus yang harus dikonfigurasi di VMWare *Player* karena program *bootloader* bekerja sebelum sistem operasi bekerja, sehingga tidak ada halangan untuk menjalankan program *bootloader*. Selanjutkan komputer virtual dinyalakan dengan *floppy drive* digunakan sebagai media boot disk-nya. BIOS akan mengenal identitas *boot sector* dengan kode 55AA, membaca *sector* tersebut ke lokasi memori 7C00 dan mengeksekusi programnya, selanjutnya program akan menampilkan keluaran tertentu sesuai dengan tujuan.

4.2. Pengkodean

Penulisan kode program dilakukan dengan cara sebagai berikut :

1. Pertama-tama membuka program *command prompt* dari Windows Start Bar (Windows XP atau Windows 7)
2. Memanggil program debug.com, dan menghasilkan prompt - (dash)
3. Mengawali penulisan program di alamat 7C00, karena di alamat inilah nanti program akan ditempatkan oleh BIOS dan mulai dieksekusi. Kode Program kemudian dapat dilihat secara lengkap seperti pada Gambar 4.

Penulisan program diawali dengan menuju ke lokasi 7C00, di sinilah di lokasi memori RAM program akan ditulis. Mengapa dipilih lokasi memori ini, karena ketika program dibaca oleh BIOS dari *bootsector*, *bootstrap* akan diletakkan di lokasi memori *offset* 7C00. Register IP (*Instruction Pointer*) otomatis akan menunjuk offset 7C00. Selanjutnya instruksi dari alamat offset 7D00-7D15 adalah rangkaian program *bootstrap*. Mulai dari alamat 7D17-7DA3 adalah lokasi data yang dinyatakan dengan db (*define byte*). Data berakhir di lokasi 7EE4, pada debug kita akan mengakhiri penulisan kode program di sini, tapi karena jumlah program belum sebesar 510 bytes, selanjutnya diarahkan ke lokasi memori 7DFE yaitu 510 desimal, di sini akan ditambahkan dua bytes angka ID yaitu

```
-A 7C00
139A:7C00 MOV AL,01
139A:7C02 MOV BH,00
139A:7C04 MOV BL,8A
139A:7C06 MOV CX,01CE
139A:7C09 MOV DX,00
139A:7C0C PUSH CS
139A:7C0D POP ES
139A:7C0E MOV AH,13
139A:7C10 MOV BP,7C17
139A:7C13 INT 10
139A:7C15 INT 20
139A:7C17 db '************************************************************',0D,0A
139A:7C59 db '*  ***  ****  **  ****  ****      ****      *** ****  *****    *',0D,0A
139A:7C9B db '*  *   ****  ****  ***  ****  ***  ***  ********  **  **    ****',0D,0A
139A:7CDD db '*     ****         **  ****      ****    *******   ***   ******',0D,0A
139A:7D1F db '*  **  ***  ******  **  ****  **    **  **********   *  ********',0D,0A
139A:7D61 db '*  ***   *  **  **  **     *  ***  ***      *******    *********',0D,0A
139A:7DA3 db '************************************************************',0D,0A
139A:7DE5

A 7DFE
DB 55AA

W 7C00 0 0 1
```

Gambar 4. Kode Program Bootloader

55AA. Selanjutnya W 7C00 0 0 1 adalah perintah pada debug.com yang artinya W adalah penulisan baris instruksi di alamat 7C00 pada drive 0 yaitu *floppy drive*, di *sector* 0 sebanyak 1 *sector*. Sehingga program sebesar 512 bytes akan ditulis ke *bootsector*. Setelah itu bootloader selesai ditransfer ke *bootsector*.

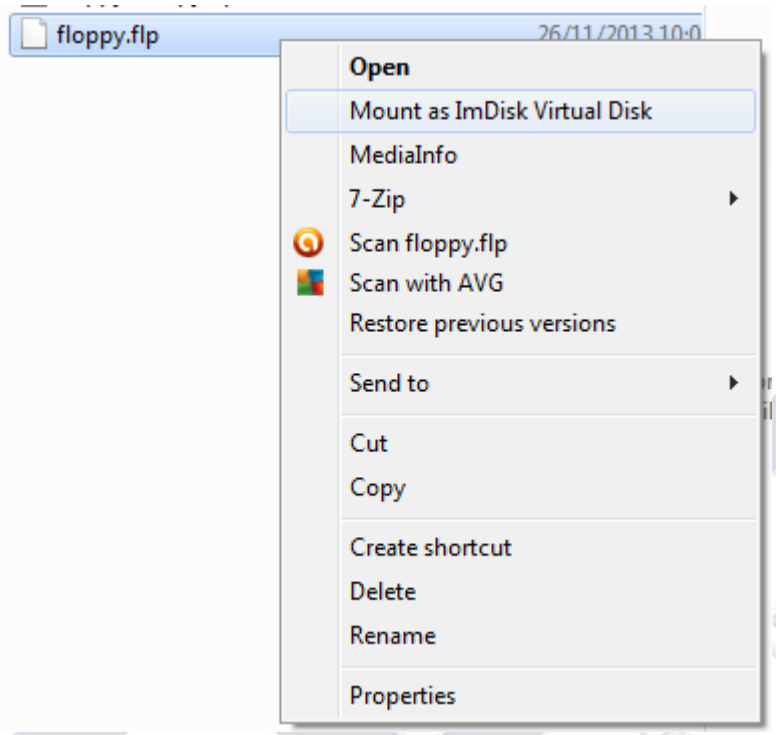
Bootstrap program pada Gambar 4 itu sendiri pada dasarnya adalah program untuk menampilkan string, menggunakan Int 10h dengan service AH=13h. Kode instruksi ini akan mencetak string yang ditunjuk oleh register BP dengan kombinasi SEGMENT:OFFSET, ES:DS. Karena program *bootstrap* hanya menggunakan satu buah *segment* maka data pada db harus diarahkan ke segment yang sama dengan CS sehingga ES perlu menyalin segment CS dengan cara PUSH CS diikuti oleh POP ES. Dengan demikian Int 10h service 13h menunjuk pada segment ES yang memiliki *segment* yang sama dengan CS. Karena kombinasi ES:BP dibutuhkan untuk meletakkan data db (*define byte*).

4.3. Ujicoba

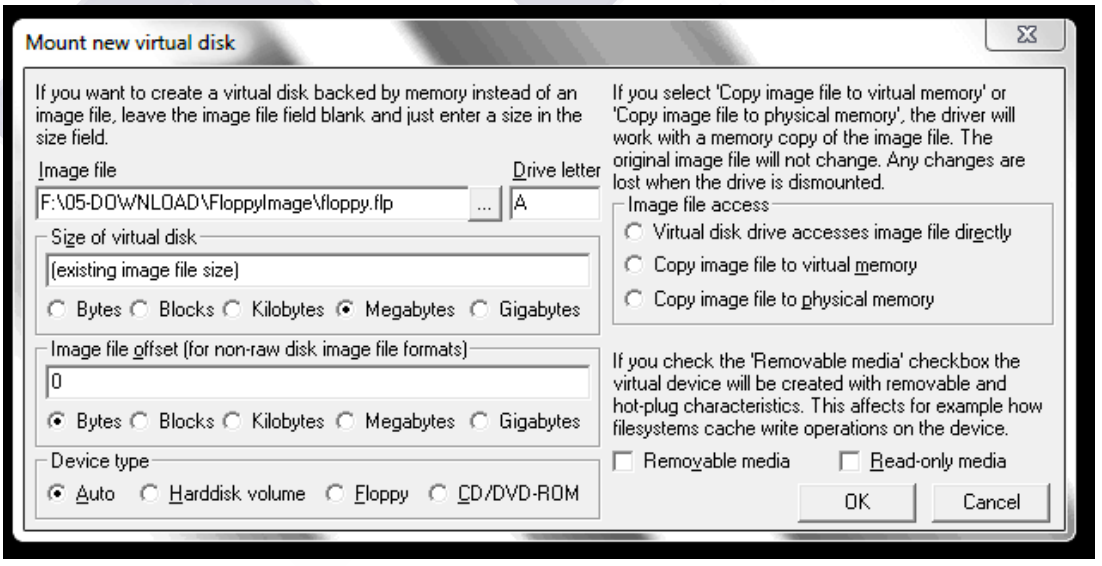
Untuk melakukan ujicoba dari hasil penulisan bootstrap ke lokasi *bootsector*, maka diperlukan *drive* agar *image floppy disk* dapat dibaca oleh sistem Windows. Untuk itu digunakan program Imdisk yang harus terlebih dahulu diinstall di Windows. Teknik *mounting image floppy* disk dapat dilakukan seperti terlihat pada Gambar 5.

Selanjutnya memilih menu *Mount as ImDisk Virtual Disk,* dan pilihan ini akan membawa pada *Windows Mount new virtual disk.*

Pada bagian *Drive letter* dipilih *drive letter* A, dengan mengarahkan *image file* di mana diletakkan *file* imagefloppy.flp. Pilihan lain dibiarkan



Gambar 5. Mounting Image Floppy Drive



Gambar 6. Mount New Virtual Disk

tetap, dan apabila proses *mounting image floppy drive* sukses maka akan diperoleh tampilan seperti pada Gambar 7.

Floppy disk drive (A) akan muncul pada bagian *removable storage* di program My Computer, pada sistem operasi Windows XP atau Windows 7. Dengan demikian kita dapat menulis program pada debug.com dan langsung mentrasfernya ke *floppy disk drive* A, selanjutnya dapat dilakukan proses *un-mounting* apabila akan masuk ke dalam proses ujicoba.

Gambar 7. Hasil Mounting Virtual Disk

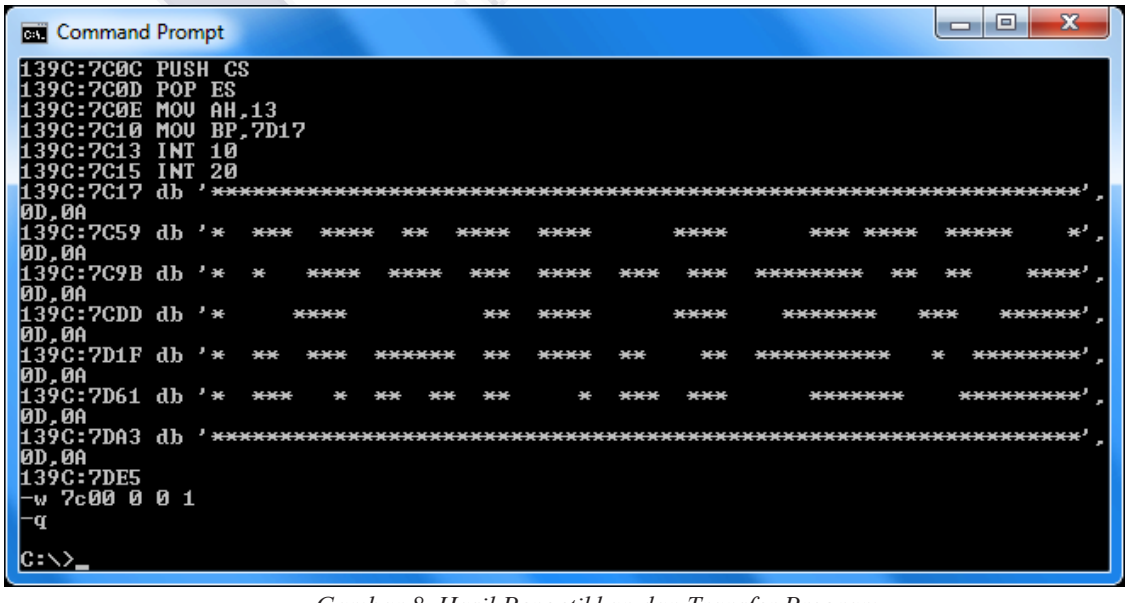
Langkah yang dilakukan cukup dengan melakukan *click* tombol kanan *mouse* dan memilih menu *un-mounting*.

Setelah proses *mouting*, Gambar 8, menunjukkan cuplikan dari proses koding yang dilakukan pada debug.com. Di akhir program terdapat perintah debug **w 7C00 0 0 1**, berarti dilakukan proses penulisan sebanyak satu *sector* dari alamat 7C00 ke drive A ke *sector boot sector*.

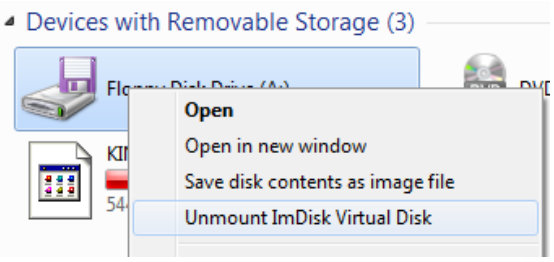
Jika sudah dilakukan proses penulisan dan sukses, selanjutnya dilakukan proses *un-mounting* seperti terlihat pada Gambar 8.

Floppy image disk yang telah di *un-mounting* selanjutnya dipasang pada program virtualisasi di VMWare Player, dan diatur proses *booting* dimulai dengan *removable disk drive* di mana di sini *floppy drive* akan dibaca terlebih dahulu sebelum membaca *harddisk*, dengan demikian program *bootstrap* yang sudah dibuat dan ditulis ke *sector boot* dari image *floppy disk* akan dibaca dan ditempatkan di lokasi memori 7C00 dan akan menghasilkan keluaran seperti pada Gambar 9.

Tampilan merupakan tanda bahwa *bootstrap* berhasil dieksekusi oleh sistem interupsi BIOS, dan juga merupakan bukti bahwa *bootstrap* berhasil ditempatkan di lokasi alamat 7C00 seperti yang direncanakan. Tampilan pada Gambar 9 merupakan sebuah tampilan kecil yang merupakan lompatan besar untuk pemanggilan program *kernel* sistem operasi yang nantinya akan menjadi program utama sistem operasi.



Gambar 8. Hasil Pengetikkan dan Transfer Program



Gambar 9. Proses Un-mount Image Floppy Disk

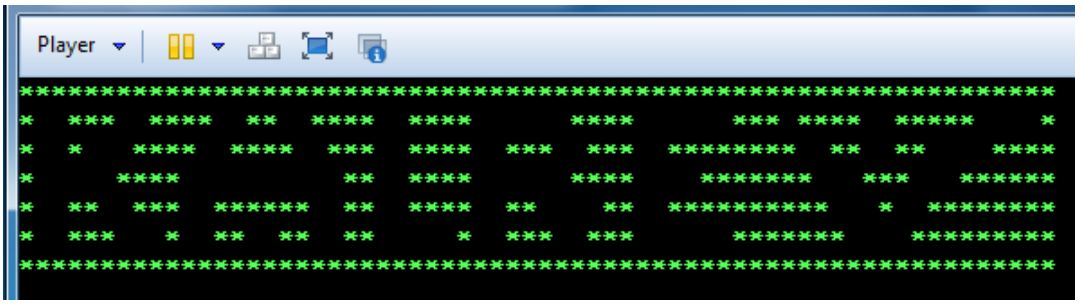
4.4. Debug

Proses pembuatan program mengalami menemui berbagai macam kesalahan yang dapat dirangkum sebagai berikut :

Karena program tidak dikompilasi tetapi langsung ditulis di memori dan ditransfer ke *boot sector* dari *floppy disk* maka perlu dilakukan beberapa kali percobaan. Alamat 7C00 adalah alamat program setelah berada di memori ketika ditempatkan oleh BIOS, sedangkan alamat pada saat pembuatan program di debug sendiri biasanya dimulai dari 100 heksadesimal. Penulisan awal dari alamat 100 biasanya dilakukan ketika program menggunakan 1 segment atau dibuat untuk dijadikan .com (dot com) program. Pada saat pembuatan program di debug. com alamat yang digunakan adalah 100 heksadesimal karena standar register IP (*Instruction Pointer*) menunjuk pada alamat 100 heksadesimal yaitu posisi di mana instruksi akan dieksekusi ketika perintah G (G0) dilakukan. Namun ketika program akan ditransfer ke *boot sector* posisi alamat awal program diletakkan di alamat 7C00 heksadesimal.

Berikutnya kesalahan yang terjadi terdapat pada alamat awal dari lokasi data yang harus disesuaikan dengan alamat awal program, karena lokasi data akan bergeser seiring dengan perubahan alamat awal. Hal ini terjadi karena data diletakkan di akhir program. Pada dasarnya lokasi data dapat diletakkan di akhir atau awal baris program, tetapi berdasarkan percobaan, lokasi data di akhir baris program menggunakan debug. com lebih menghasilkan data yang stabil. Ketidakstabilan program debug dapat terjadi apabila terjadi nilai-nilai register yang berubah karena terjadinya eksekusi instruksi-instruksi. Untuk mengatasi hal ini perlu dilakukan exit dari program debug.com dan kemudian masuk kembali ke dalam program.

Jumlah data yang dilampirkan totalnya tidak boleh lebih dari 510 bytes, karena masih ada 2 byte data sebagai pengenal *boot sector*. Karena data dan



Gambar 10. Hasil Keluaran Program Bootstrap Bootloader

program berada di dalam lokasi segment yang sama perlu dilakukan beberapa kali ujicoba untuk melihat jumlah data yang pas dengan cara melakukan perhitungan panjang program atau dapat juga dengan menuliskannya langsung di debug sehingga dapat diketahui posisi alamat akhir *offset* program.

Hal penting lain yang perlu diperhatikan adalah proses penulisan program ke lokasi *boot sector* dari *floppy disk* perlu diperhitungkan dengan tepat karena kesalahan peletakkan program di *sector* akan menyebabkan program *bootstrap* tersebut tidak dapat dibaca oleh BIOS, atau BIOS tidak dapat menemukan ID yang tepat di *boot sector*.

4.5. Analisis Hasil

Program *bootstrap bootloader* yang sudah dibuat pada dasarnya adalah program yang dapat menampilkan suatu keluaran di layar komputer. Yang menjadi kesulitan adalah bahwa program ini berjalan di tahap awal bekerjanya sistem komputer sehingga tidak ada *library* pemrograman yang dapat digunakan untuk membuat program yang biasanya diperoleh dari *compiler* yang bekerja di atas sistem operasi. Program *bootstrap bootloader* ini bekerja secara langsung ke *hardware* (*direct programming*) dengan demikian *library* yang dapat digunakan hanyalah *library* sistem interupsi BIOS. Sistem interupsi BIOS sendiri hanya dapat digunakan di mode real prosesor. Prosesor x86 kompatibel dapat beroperasi pada dua mode yaitu *mode real* dan *mode protected*. Dalam kasus ini Intel Corp selaku produsen prosesor x86 menjaga kompatibilitas program-program terdahulu sehingga dapat bekerja pada mode real, sehingga pada awal inisiasi sistem komputer, prosesor selalu masuk ke dalam mode real. Di sinilah kesempatan program *bootloader* menggunakan sistem interupsi yang tersedia di *mode real*. Selanjutnya adalah bagaimana program yang hanya dapat ditempatkan pada sebuah sector yaitu sebesar 512 byte dapat digunakan sebagai jembatan atau proses antara sebelum memanggil program kernel berikutnya.

Dalam sistem *library* yang ada proses pembacaan program di luar satu *sector* hanya dapat dilakukan menggunakan interupsi, jika akan menggunakan program yang besarnya membutuhkan lebih dari satu *sector*, maka program tersebut harus ditulis secara manual ke masing-masing *sector* dari media *boot* atau penyimpanan. Teknik ini tentu saja membuat program *bootloader* dan sistem operasi tidak dapat didistribusikan dengan mudah. Selanjutnya diperlukan pembuatan *file system* untuk memungkinkan menyimpan program yang lebih besar dari satu sector sehingga program dan data yang dibuat tidak ditulis secara manual ke sector-sector tetapi disimpan dalam file / berkas digital.

Program *bootstrap loader* pada

gilirannya hanya akan menjadi tahap awal dari dua tahap proses pemanggilan *kernel* yang menjadi tugas utama dari *bootloader*. Tahap awal adalah membuat program pemanggil program lain yang ditempatkan di luar *boot sector* dengan cara menggunakan *file system*. Artinya *bootloader* berisi *bootstrap* yang berfungsi untuk membuat dan mengenal *file system* dan mampu memanggil file *program kernel*.

Hasil yang sudah diperoleh dengan hanya menampilkan keluaran sederhana yang disimpan di *sector* yang sama bersama dengan program *bootstrap*-nya tentu saja merupakan tahap antara yang bukan menjadi tujuan akhir dari penelitian ini. Tapi pemahaman yang dalam akan tahap ini menjadi suatu hal yang sangat penting untuk mengembangkan program *bootloader* menjadi program yang lebih lengkap yaitu mampu mengenal *file system* dan memanggil program kernel sistem operasi.

Satu hal khusus yang belum pernah atau jarang dibahas dalam beberapa penulisan program *bootloader* adalah penggunaan program debug.com untuk melakukan penulisan kode-kode program langsung ke memori RAM dan menuliskannya langsung ke bagian *boot sector* dari *boot device*, merupakan cara yang praktis dan mudah, terutama untuk digunakan sebagai teknik sederhana untuk mengajarkan prinsip sistem komputer dan proses *booting* sistem operasi tanpa harus mempelajari bahasa *assembly* dulu kepada setiap pemula dan peminat pengembang sistem operasi.

V. Penutup

5.1. Kesimpulan

Dari hasil penelitian untuk penulisan jurnal ini ditemukran beberapa hal yang dapat disimpulkan sebagai berikut :

1. Pembuatan kode-kode program langsung di memori dengan menggunakan program debug.com dapat dimanfaatkan untuk menulis program *bootloader* langsung di alamat dan *sector boot device*.
2. Program *bootloader* yang hanya sebesar 512 bytes dapat dikembangkan untuk menjalankan data yang berada di luar *boot sector* dengan menggunakan sistem interupsi BIOS.
3. Penggunaan lingkungan virtual menggunakan VMWare player sangat memudahkan proses pengembangan program *bootloader* dibandingkan harus menggunakan komputer fisik.
4. Program *bootloader* yang sudah dibuat masih merupakan pengembangan tahap awal, dibutuhkan penelitian lebih lanjut untuk pengembangan sampai ke tahap pemanggilan *kernel* sistem operasi.

5. Penelitian pengembangan program *bootloader* sangat penting dilakukan untuk memahami cara kerja sistem komputer dan sistem operasi sebagai dasar lebih lanjut untuk penelitian sistem operasi.

5.2. Saran

1. Pengembangan lebih lanjut dari pembuatan program *bootloader* dapat menggunakan *compiler* bahasa *assembly*, ketika program yang dibuat sudah semakin rumit.
2. Perlu dipertimbangan alternatif pembuatan program *bootloader* untuk mode proteksi atau 32 bit, karena memiliki banyak keunggulan.
3. Program *bootloader* yang dibuat disiapkan untuk menggunakan *File Allocation Table* (FAT), atau sistem file lainnya.

DAFTAR PUSTAKA

[1] Silberschatz, Galvin, Gagne. 2011. Operating System Concepts. International Student version. Wiley. Eight Edition.

[2] Mchoes, Flyn. 2011. Understanding Operating System. Course Technology Cengage Learning. International Edition. Six Edition.

[3] Lukito, Ediman. 1990. Dasar-Dasar Pemrograman dengan Assembler 8088. Elex Media Komputindo.

[4] Stallings, William. 2012. Operating Systems Internal and Design Principles. Seventh Edition. Pearson.

Internet

[5] Mike, 2009, Operating System Development, http://www.brokenthorn.com/Resources/OSDev0.html, diakses pada tanggal 2 September 2013.