LAPORAN PRAKTIKUM

MODUL 1

PENGENALAN SITEM PENGEMBANGAN OS DENGAN PC SIMULATOR ‘BOSCH’

PRAKTIKUM SISTEM OPERASI

OLEH :

MUHAMMAD ASHARUL MAALI

L200190128

PRAKTIKUM SO D

INFORMATIKA

FAKULTAS KOMUNIKASI DAN INFORMATIKA

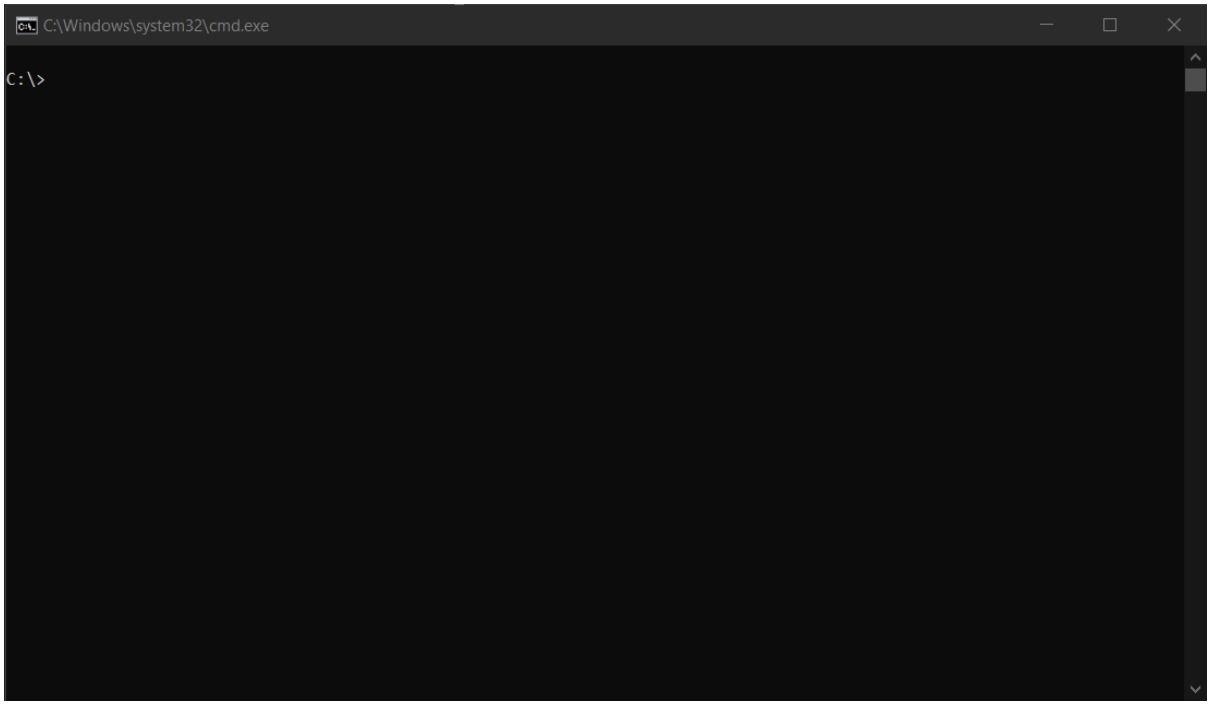
UNIVERSITAS MUHAMMADIYAH SURAKARTA

2020

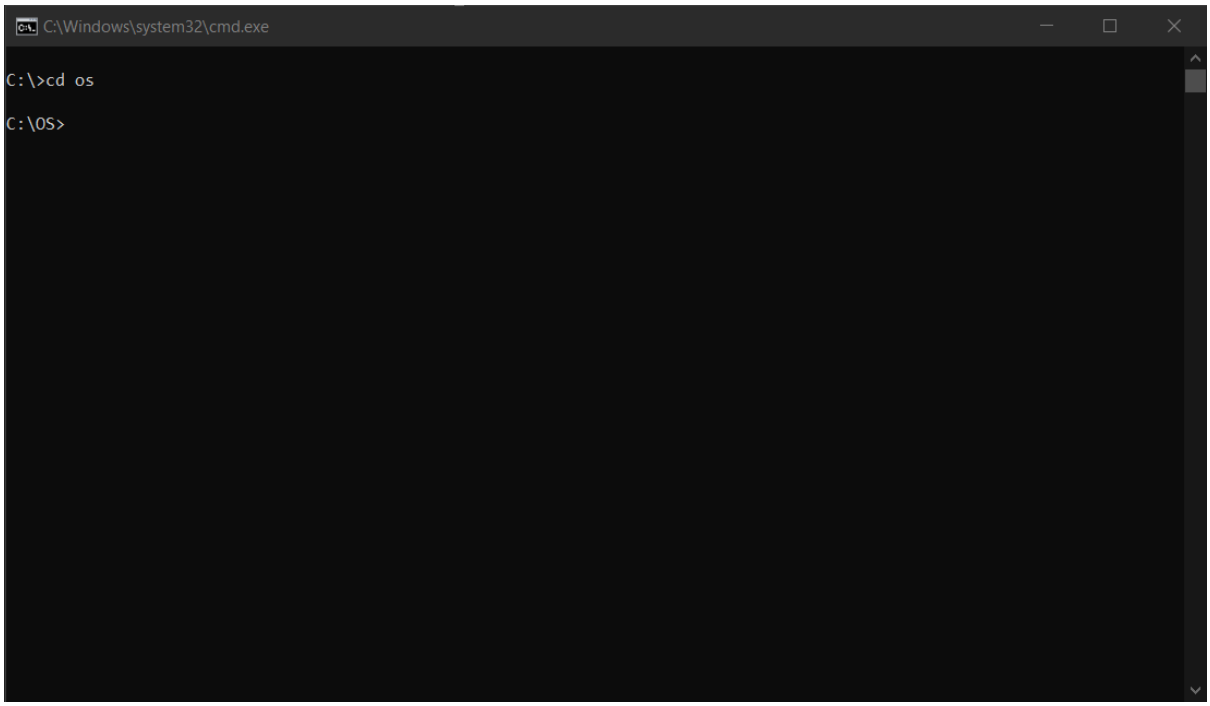
Langkah Kyerja

A. **Menuju ke direktori kerja**.

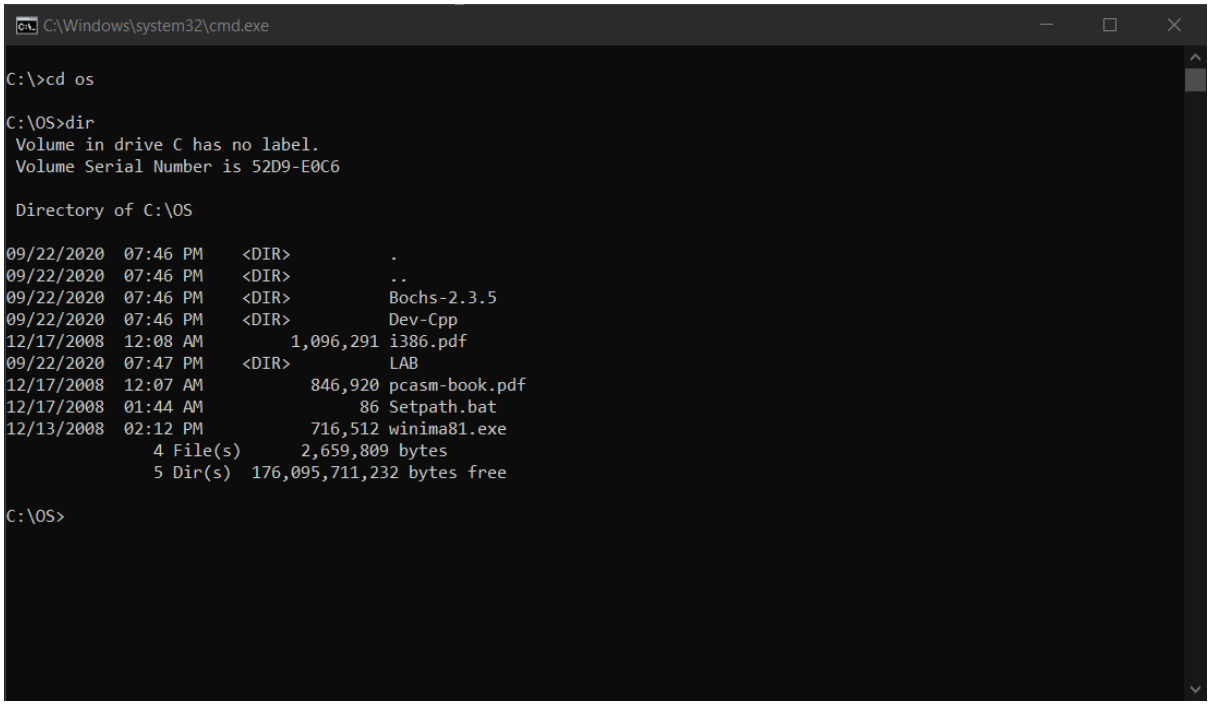
a. Jalankan program command promt atau cmd.



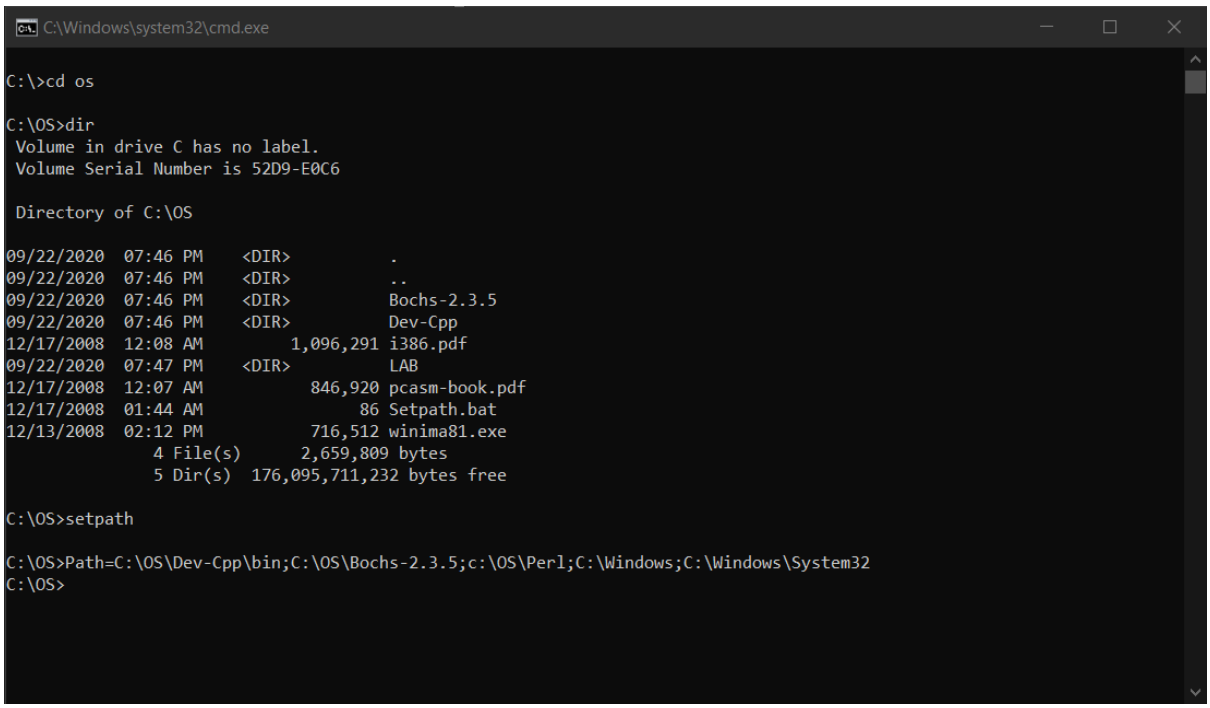
b. Masuk ke direktori kerja ’C:\OS’, dengan perintah ’cd os’ .

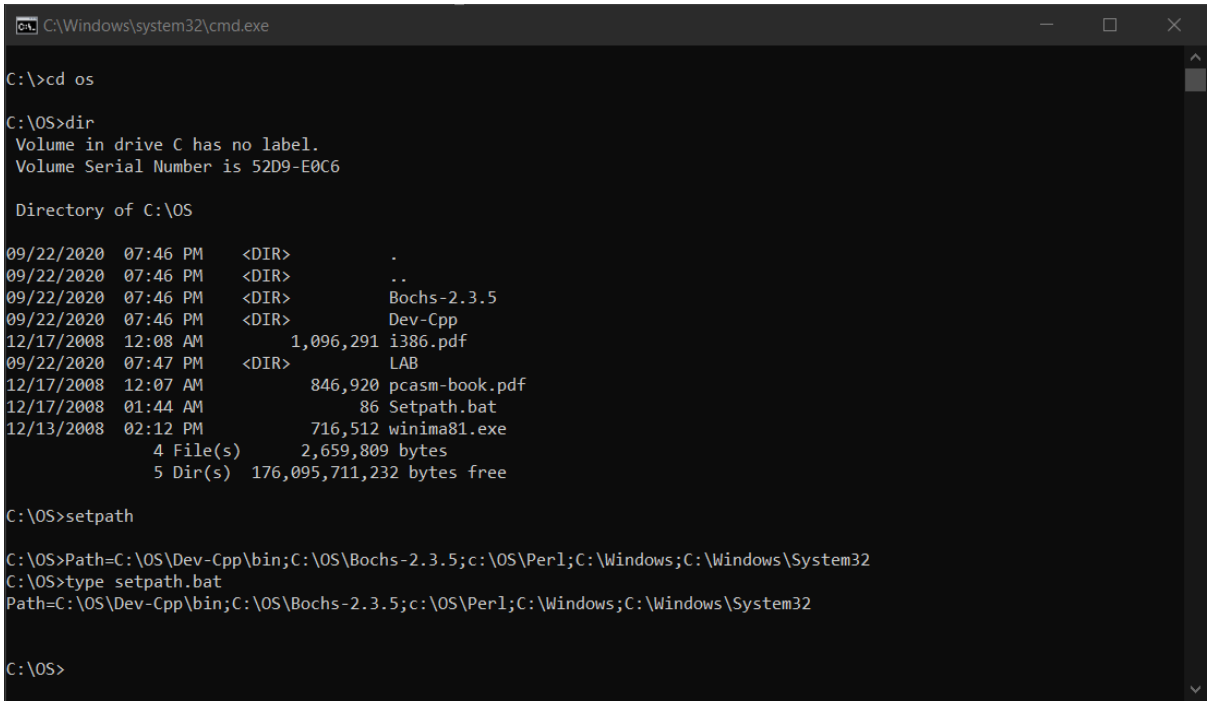


c. Masukan perintah dir, untuk melihat isi direktori di dalam folder tersebut. Akan muncul seperti di tampilkan pada gambar



d. Jalankan file setpath, untuk menjalankannya ketik ‘setpath’ tekan File ‘setpath.bat’ digunakan untuk mengatur lingkungan kerja (‘path’) selama anda melakukan praktikum, anda harus menjalankan progam ini sebelum memulai setiap sesi praktikum anda, untuk menjalakannya ketik ‘setpath’ tekan . Perhatikan teks yang muncul di layar, seperti ditampilkan pada Gambar 1.1. Untuk melihat script yang terdapat di dalam file ‘setpath.bat’ dapat digunakan perintah ‘type setpath. bat’, cobalah. ‘PATH’ dapat bersisi banyak daftar lokasi, di antara item lokasi di batasi dengan karakter ‘;’ (titik koma). Variabel ‘PATH’ ini akan digunakan oleh windows untuk mencari lokasi file yang dipanggil oleh ‘user’. Urutan pencarian dimulai dari daftar lokasi yang paling awal.



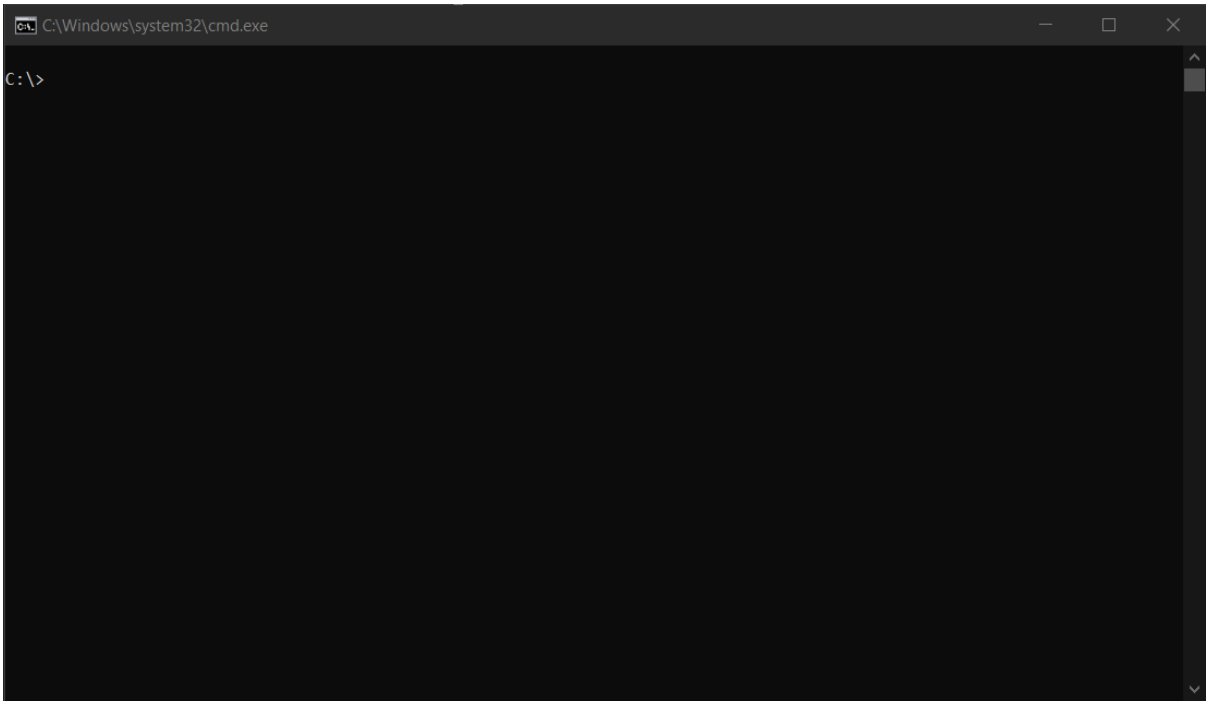


e. Untuk pengaturan ‘path’ seperti ditampilkan pada gambar 1.2 ., maka ketika user memasukan perintah (seperti memangil program) pertama kali yang dilakukan Windows adalah mencari file program di dalam direktori kerja saat itu, jika di sana tidak ditemukan file yang dimaksud, selanjutnya Windows akan meneruskan pencarian file di lokasi yang terdaftar pada variabel ‘PATH’ dimulai dari lokasi ‘C:\OS\DevCpp\bin’ jika file belum ditemukan pencarian dilanjutkan ke lokasi ‘C:\Windows’, pencarian terus dilakukan sampai ke lokasi terakhir ‘C:\Windows\system32’. Jika sampai lokasi terakhir file tidak juga ditemukan maka windows akan manmpilkan informasi bahwa file yang dimaksud tidak ada di dalam sistem. Sebaliknya jika file sudah ditemukan maka windows akan menghentikan pencarian dilokasi terakhir ditemukannya file yang dimaksud. Selain file pengatur lingkungan kerja, pada direktori kerja-anda juga terdapat tiga folder yaitu folder ‘Bochs-2.3.5’, merupakan lokasi dari program ‘PC-Simulator’ yang akan digunakan untuk membuat PC-virtual, untuk menjalankan program ‘bootstrap loader’ dan program-program yang anda buat selama menjalankan praktikum. Folder ‘Dev-Cpp’ berisi file program kompiler ‘nasm’ dan ‘gcc’ yang dapat digunakan untuk mengkompilasi program yang ditulis dalam bahasa c, c++ maupun bahasa assembly, selain itu pada folder ini juga terdapat beberapa program bantu, seperti ‘dd.exe’ untuk menyalin byte data dari satu file ke file yang lainnya. File-file kerja selama melakukan praktikum diletakan di bawah folder ‘Lab’.

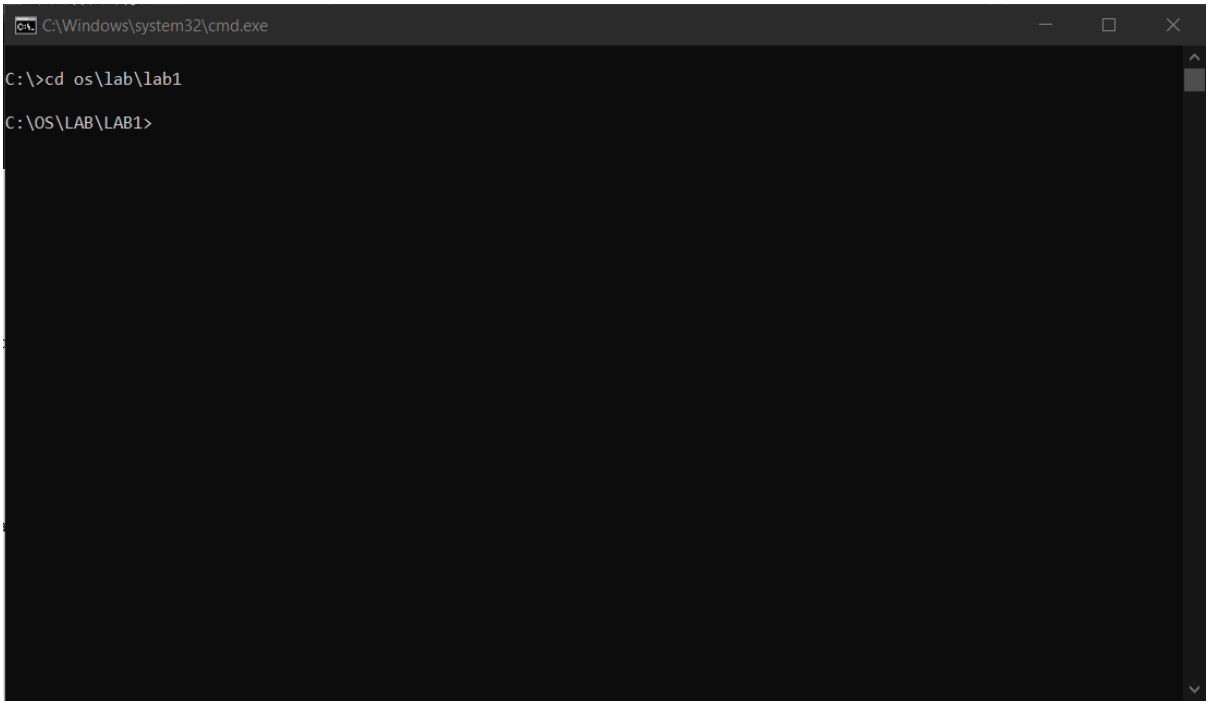
B. Melihat isi direktori kerja

Untuk masuk diirektori kerja untuk modul ini pertama adalah

a. Jalankan command prompt

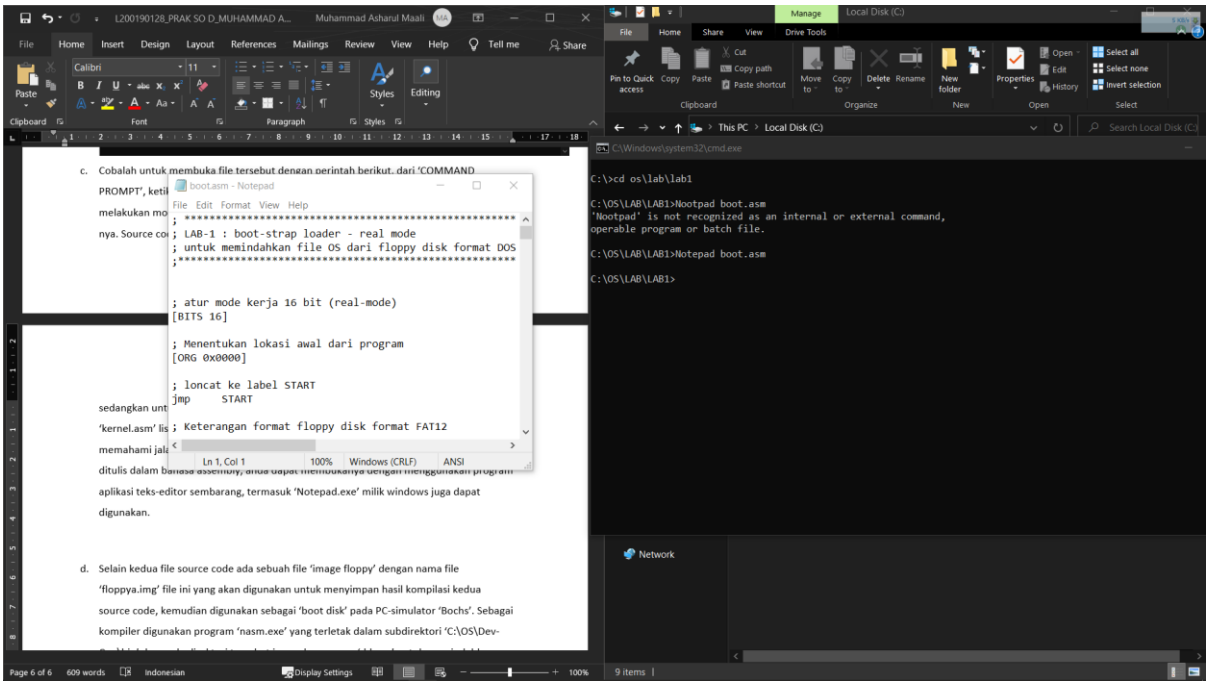


b. Masuk ke direktori kerja pada ‘C:\OS\LAB\LAB1’.



c. Cobalah untuk membuka file tersebut dengan perintah berikut, dari ‘COMMAND PROMPT’, ketikkan ‘Notepad boot.asm’ dan tekan . Saat ini sebaiknya anda tidak melakukan modifikasi terhadap program tersebut. Tutuplah kembali program ‘notepad’-nya. Source code prototype program bootloader disimpan dalam file ‘boot.asm’

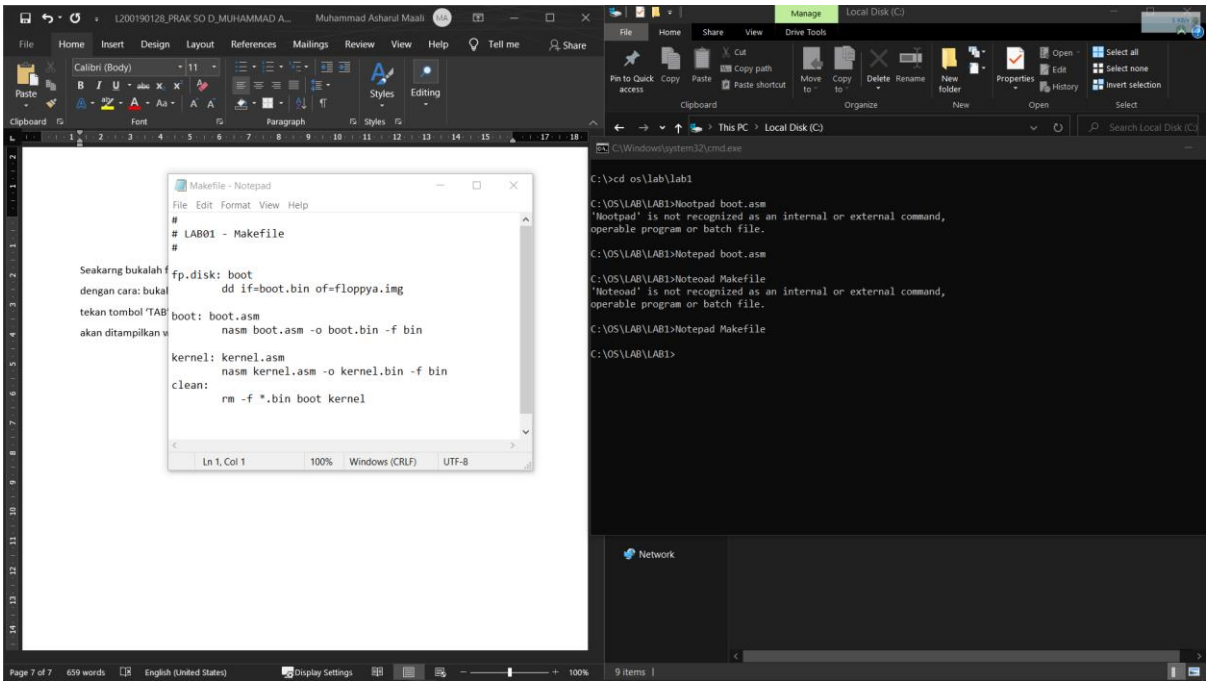
sedangkan untuk source code prototype program kernel disimpan dalam file ‘kernel.asm’ listing lengkap keduanya bisa dibaca di lampiran. Sebaiknya anda memahami jalannya kedua program sebelum melakukan praktikum. Kedua program ditulis dalam bahasa assembly, anda dapat membukanya dengan menggunakan program aplikasi teks-editor sembarang, termasuk ‘Notepad.exe’ milik windows juga dapat digunakan.



d. Selain kedua file source code ada sebuah file ‘image floppy’ dengan nama file ‘floppya.img’ file ini yang akan digunakan untuk menyimpan hasil kompilasi kedua source code, kemudian digunakan sebagai ‘boot disk’ pada PC-simulator ‘Bochs’. Sebagai kompiler digunakan program ‘nasm.exe’ yang terletak dalam subdirektori ‘C:\OS\Dev-Cpp\bin’ dan pada direktori tersebut juga ada program ‘dd.exe’ untuk memindahkan byte data dari satu file ke file yang lain, program ‘make.exe’ untuk mempercepat proses kompilasi. Pada saat ini kita akan menggunakan ketiga program tersebut yang dikemas dalam script ‘Makefile’ sehingga proses kompilasi menjadi lebih cepat, cukup dengan memanggil ‘make’. Script ‘makefile’ berupa file dengan format teks yang

C. Sekilas tentang Makefile

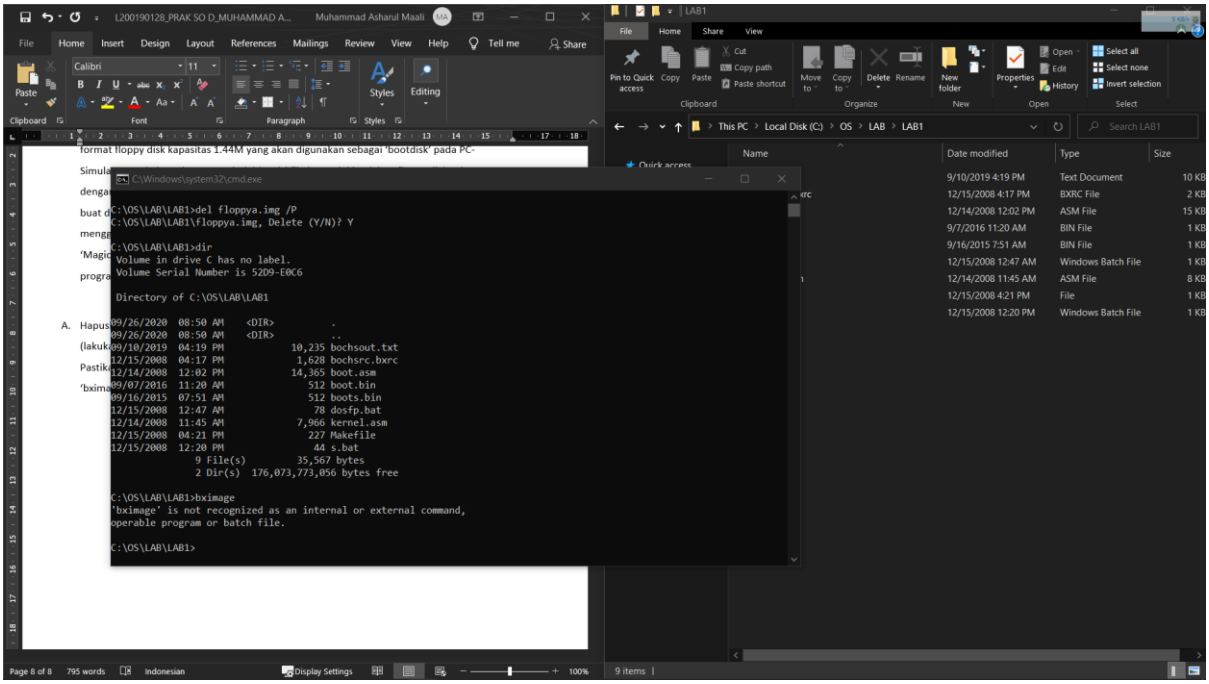
Seakarng bukalah file ‘Makefile’, dari ‘Command Prompt’ untuk mengetahui script makefile dengan cara: bukalah direktori kerja anda ‘C:\OS\LAB\LAB1’ selanjutnya ketik ‘Notepad M’ tekan tombol ‘TAB’ sehingga muncul ‘Notepad Makefile’ dan tekan . Jika langkah anda benar akan ditampilkan windows ‘Notepad’ dengan file ‘Makefile’ yang siap disunting

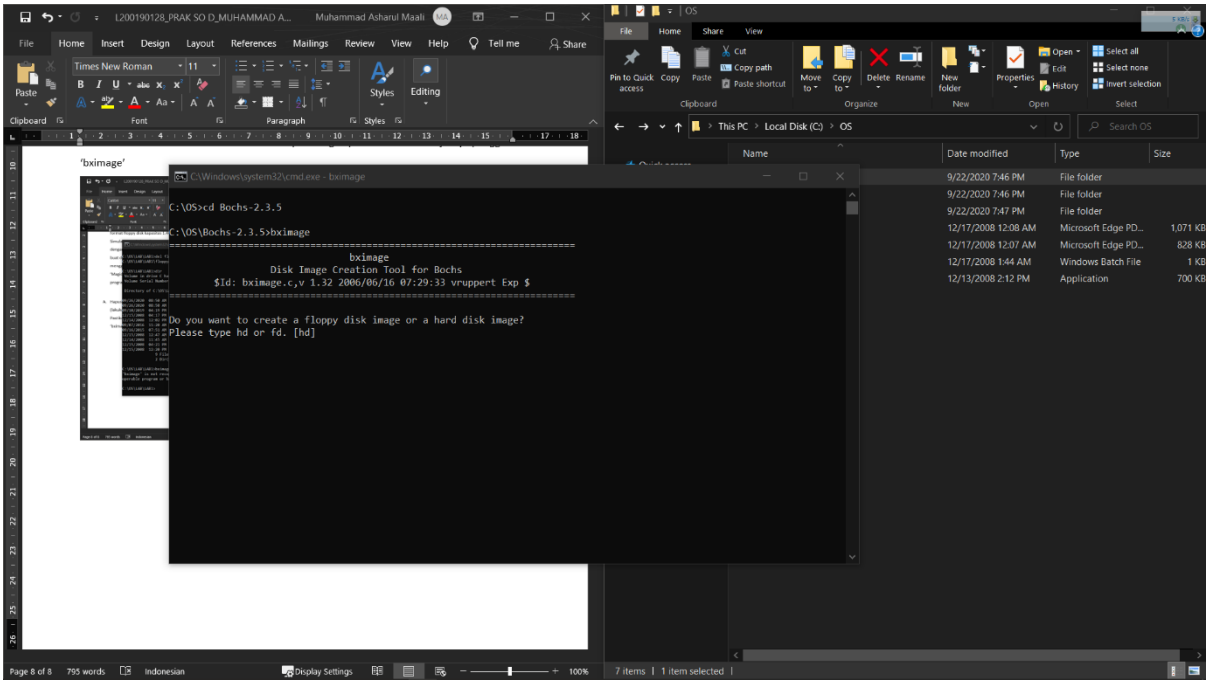


D. Mengenal Boot Disk

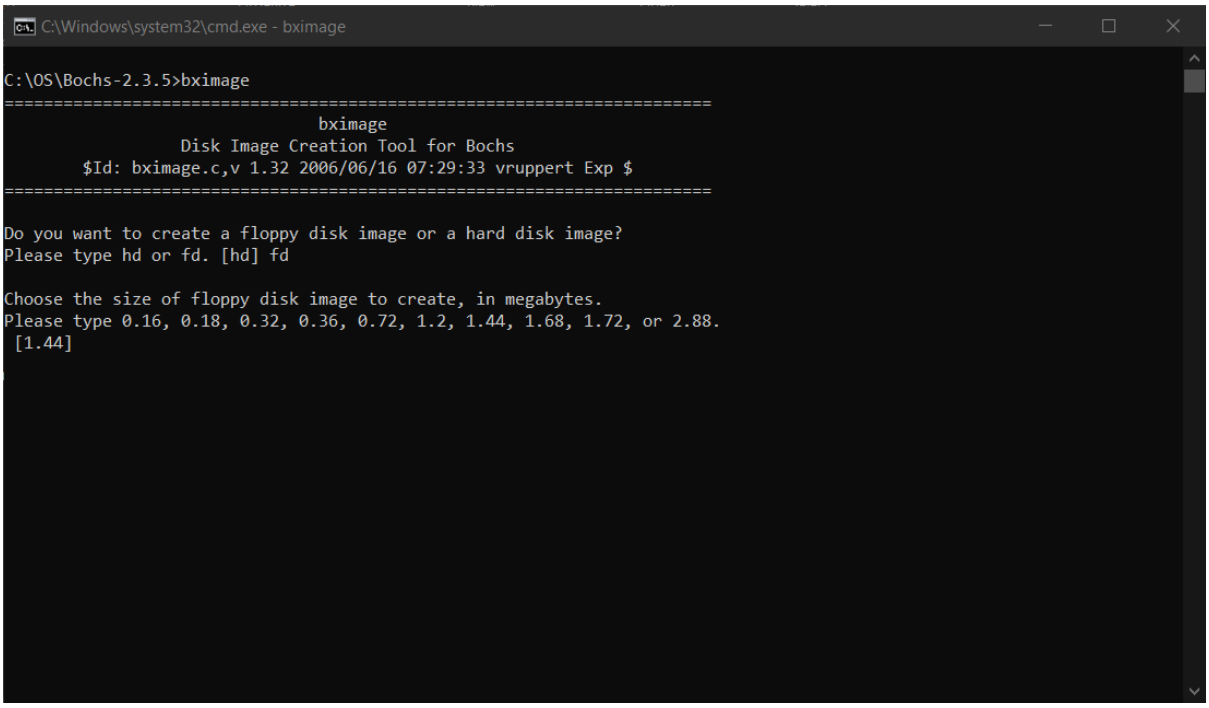
File ‘floppya.img’ adalah file image (bukan gambar), sebuah file yang isinya diformat seperti format floppy disk kapasitas 1.44M yang akan digunakan sebagai ‘bootdisk’ pada PC-Simulator, anda juga dapat memindahkan isi file image ini ke dalam floppy disk sebenarnya dengan menggunakan program bantu ‘rawrite’ atau yang lainnya. File image ini dapat di buat dengan menggunakan progam bantu bawaan ‘Bochs’ yaitu ‘bximage’ atau menggunakan program aplikasi ‘Virtual PC’ milik Windows (free juga) atau dengan program ‘Magiciso’. Sekarang kita coba membuat file image floppy baru dengan menggunakan program aplikasi ‘bximage.exe’, lakukan urutan perintah berikut.

A. Hapuslah file ‘floppya.img’ jika sudah ada pada direktori kerja anda, dari ‘Command Prompt’ (lakukan dari direktori kerja) ketik ‘del floppya.img /P’ lanjutkan dengan tekan ‘Y’ dan . Pastikan bahwa file sudah benar- benar terhapus dengan perintah ‘dir’. Selanjutnya panggil ‘bximage’

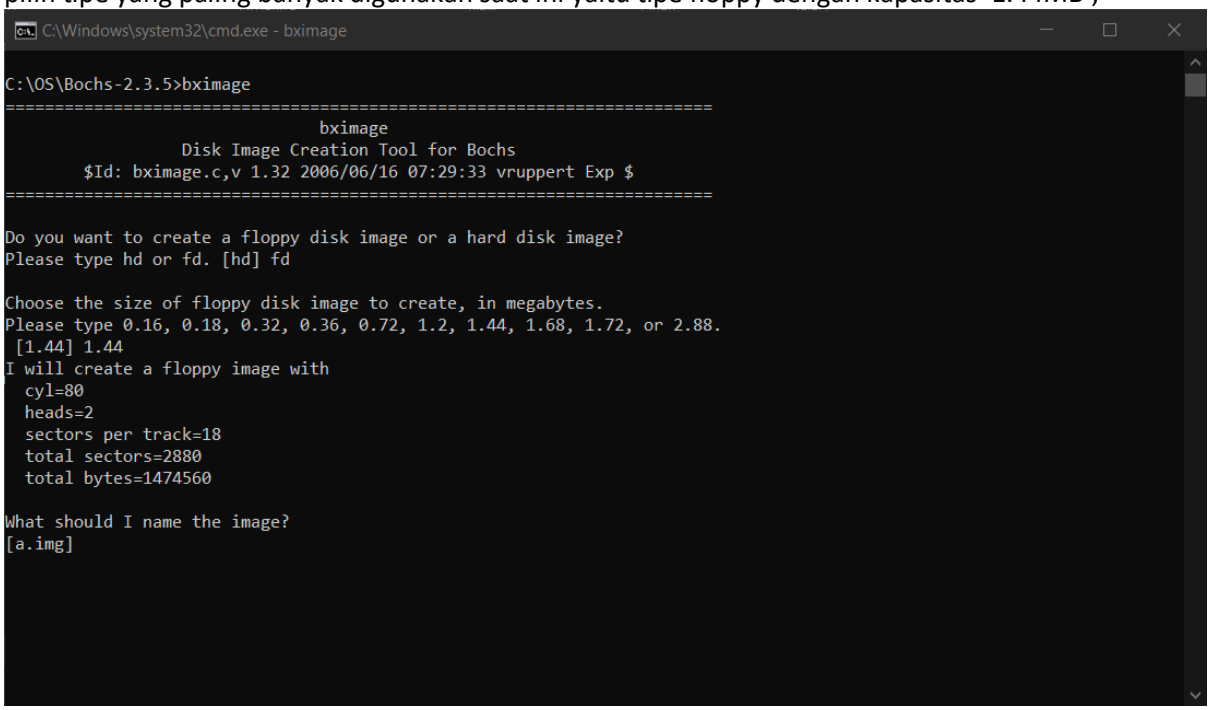




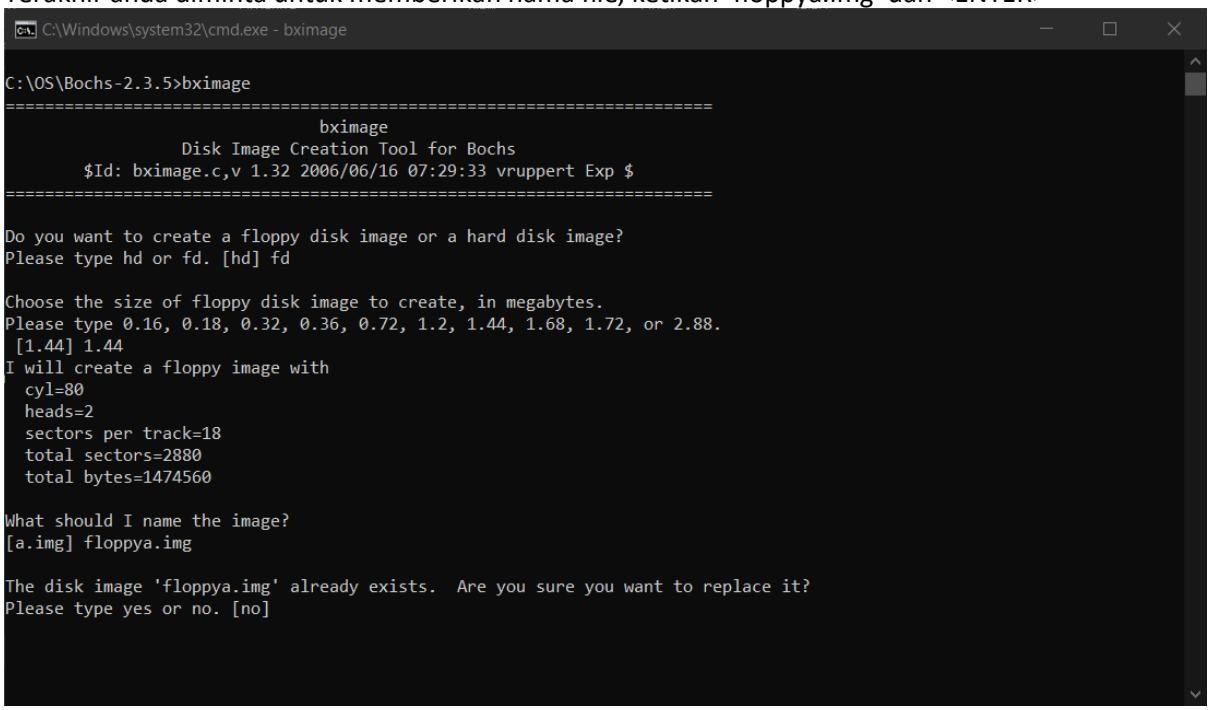
B. ketikan ‘fd’ dan tekan <Enter>

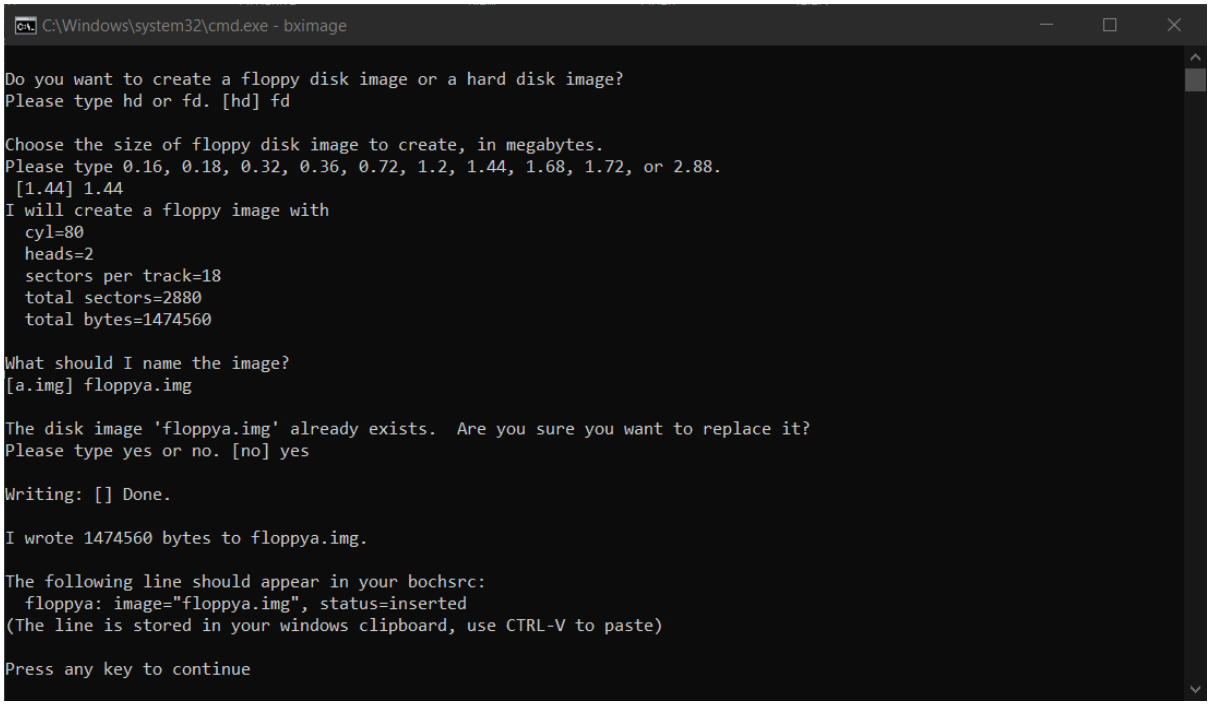


C. pilih tipe yang paling banyak digunakan saat ini yaitu tipe floppy dengan kapasitas ‘1.44MB’,

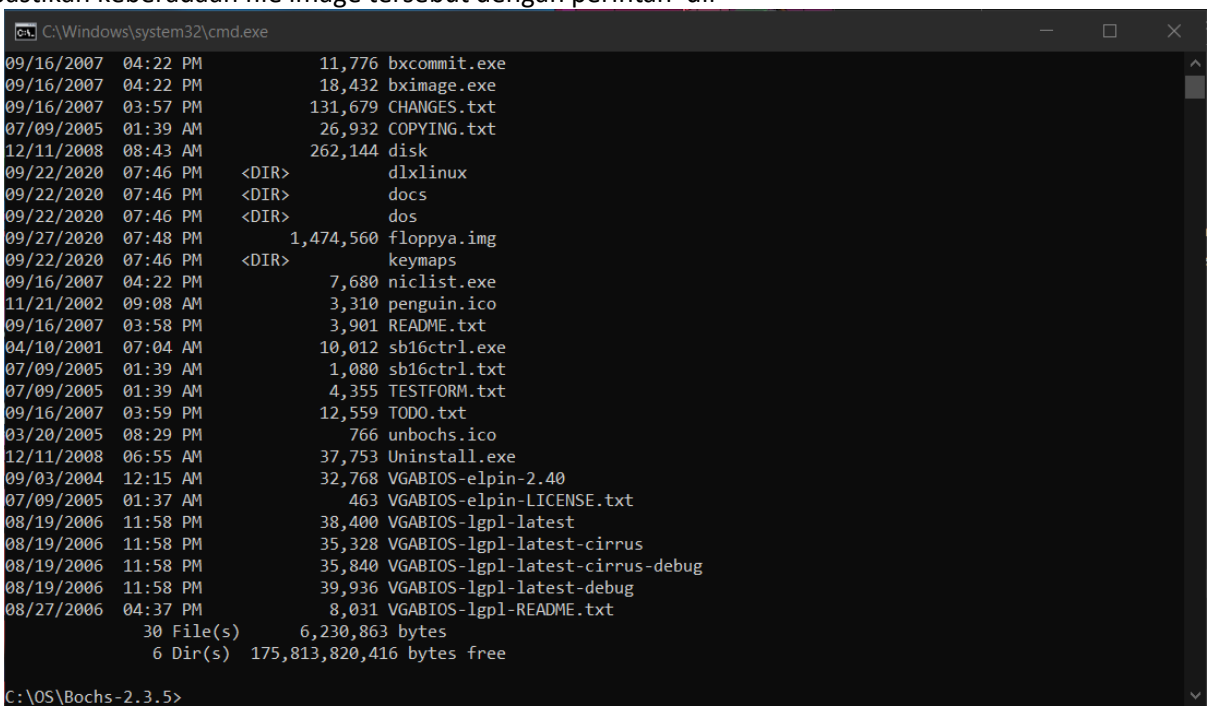


D. Terakhir anda diminta untuk memberikan nama file, ketikan ‘floppya.img’ dan <ENTER>

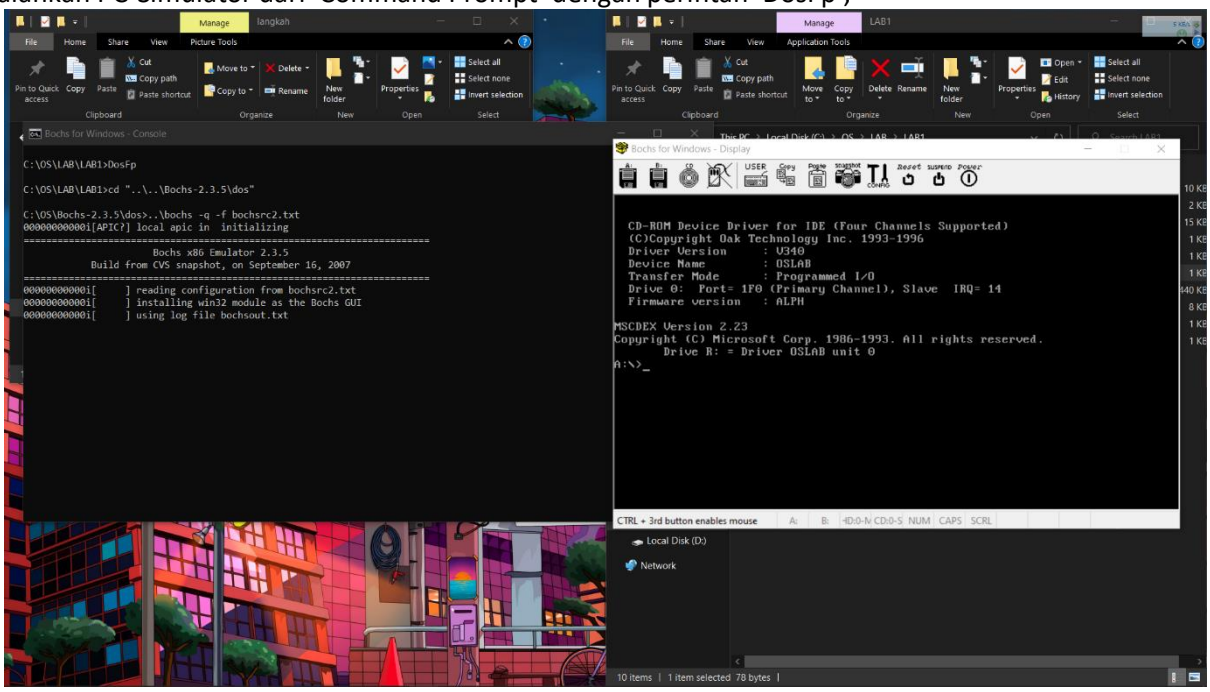




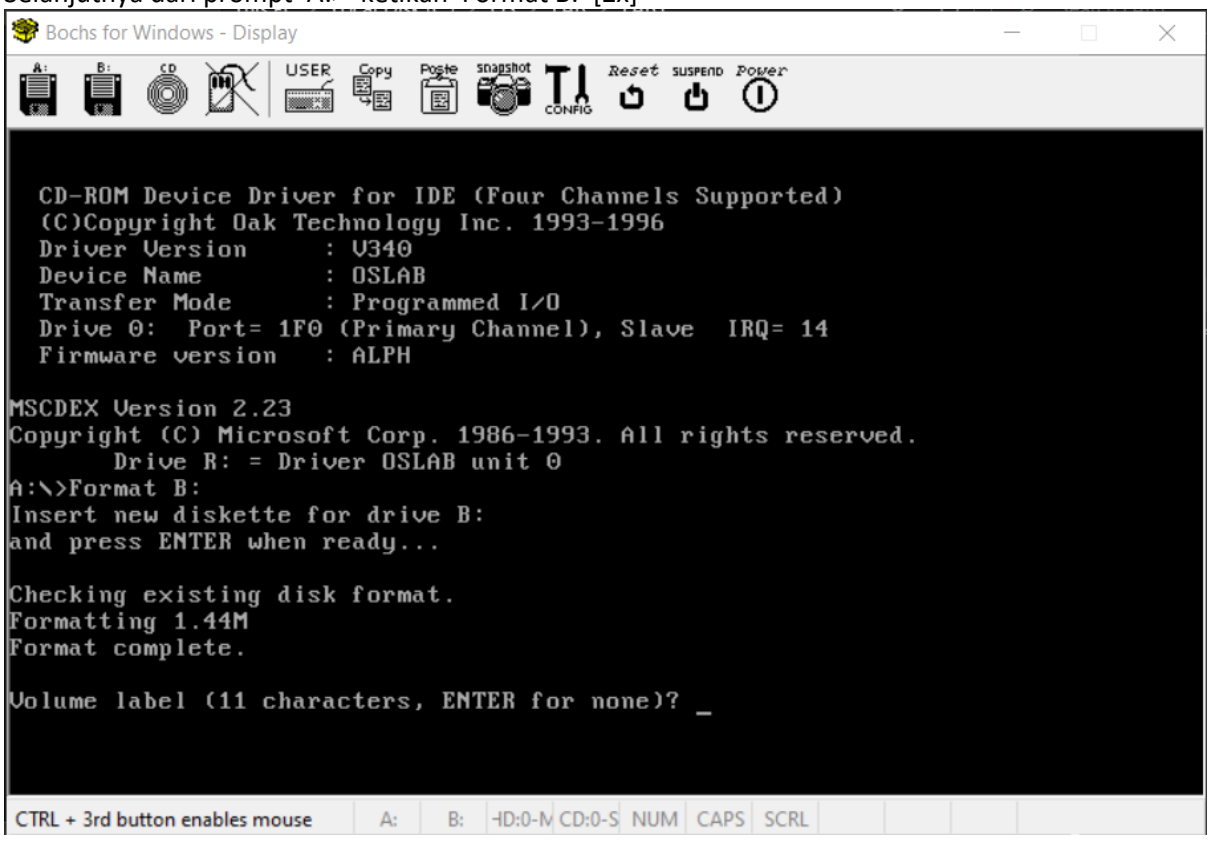
E. pastikan keberadaan file image tersebut dengan perintah ‘dir’



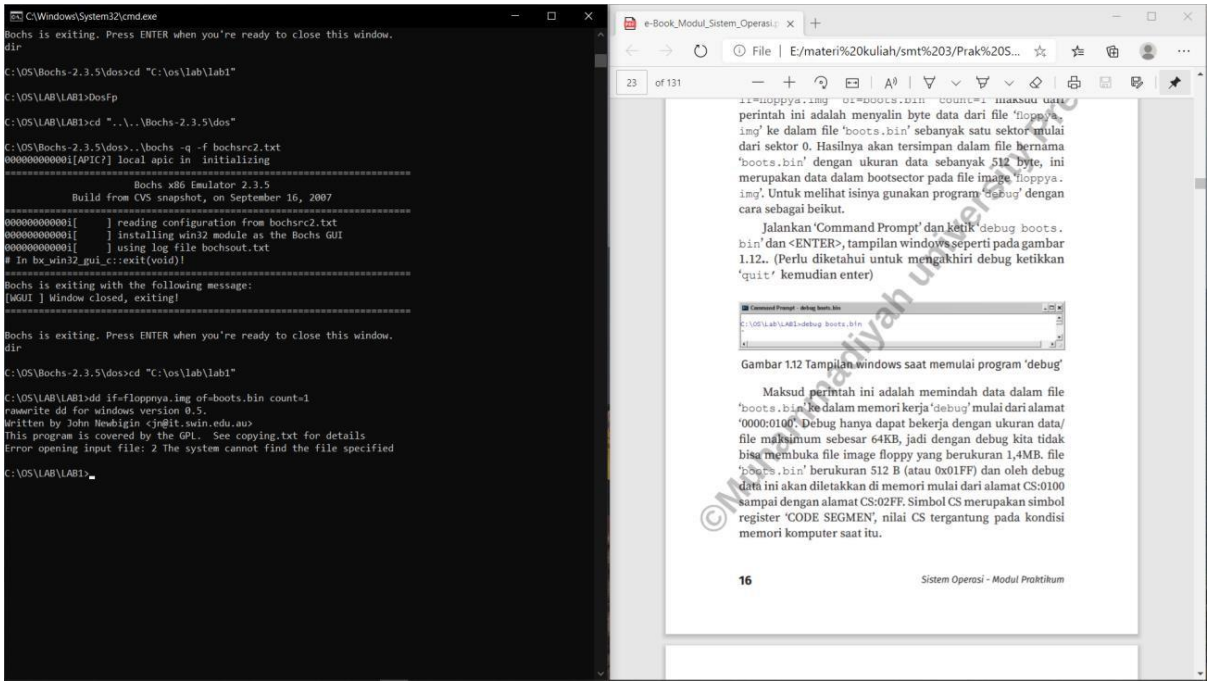
F. jalankan PC-Simulator dari ‘Command Prompt’ dengan perintah ‘DosFp’,



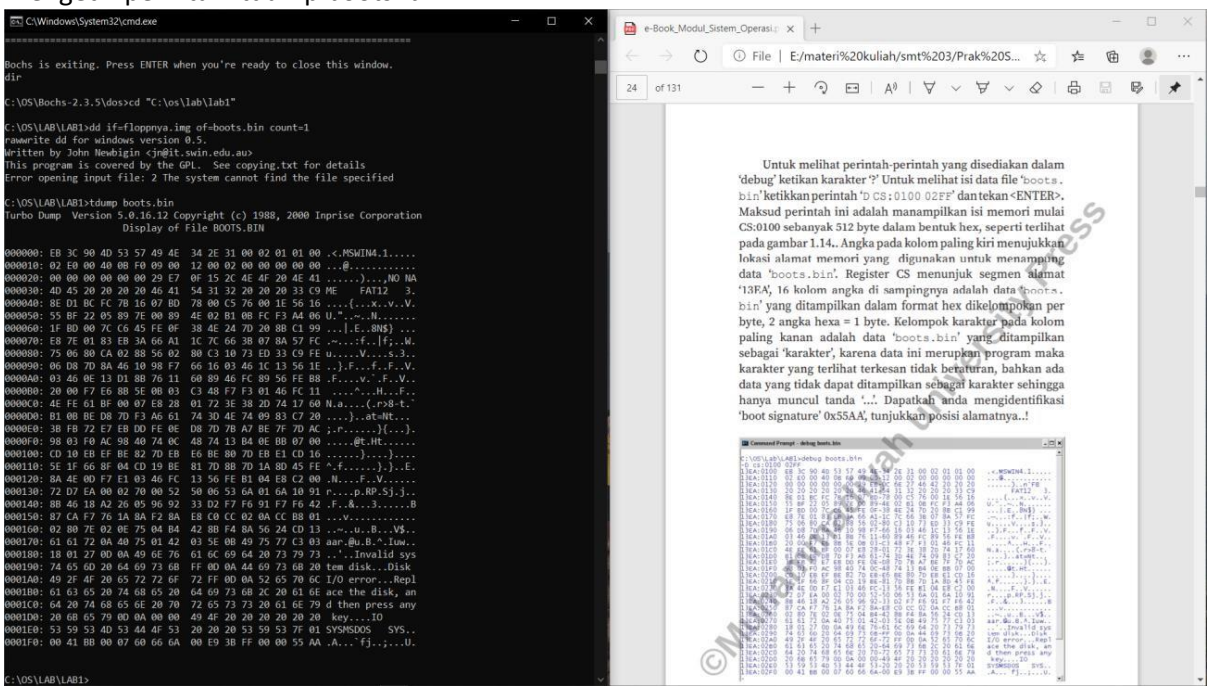
G. Selanjutnya dari prompt ‘A:>’ ketikan ‘Format B:’ [2x]



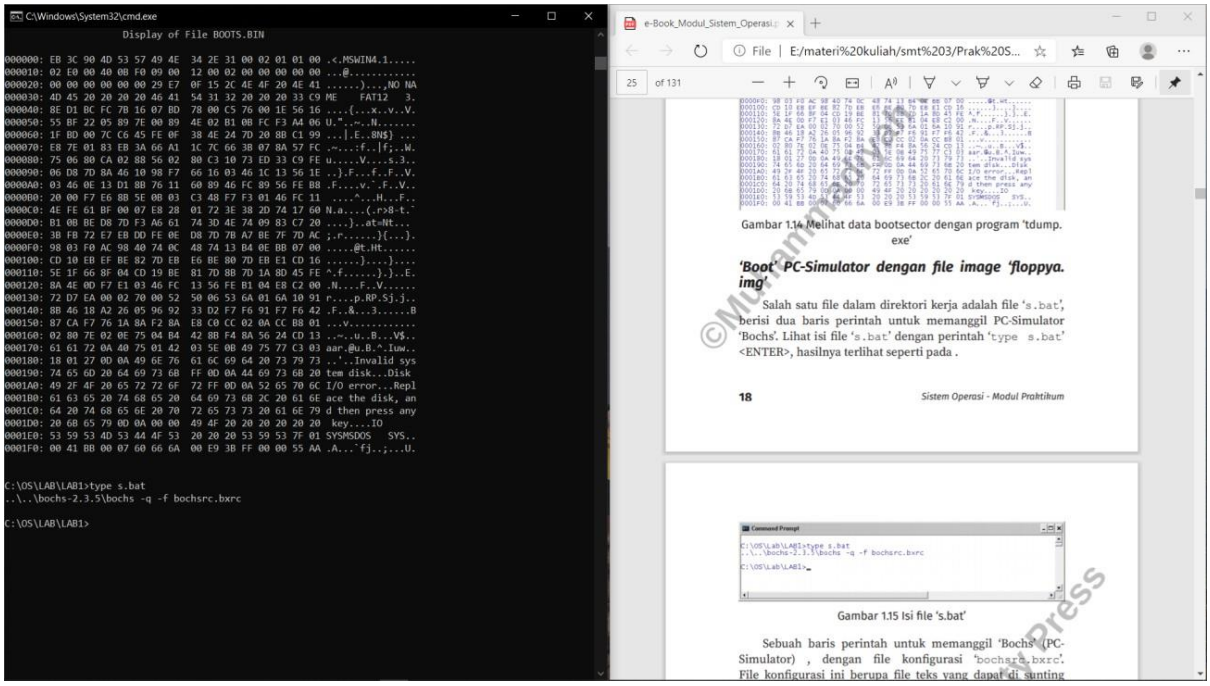
H. Tutup kembali PC-Simulator dengan klik pada tombol power, di bagian menu atas-kanan. Sekarang ‘floppya. img’ sudah terformat dan dapat digunakan untuk menyimpan data, namun belum dapat digunakan untuk ‘booting’, karena pada ‘bootsector’ belum diisi program ‘bootloader’ dan Copy 512 byte data bootsektor ke dalam sebuah file terpisah, caranya dari ‘Command Prompt’ ketik ‘dd if=floppya.img of=boots.bin count=1



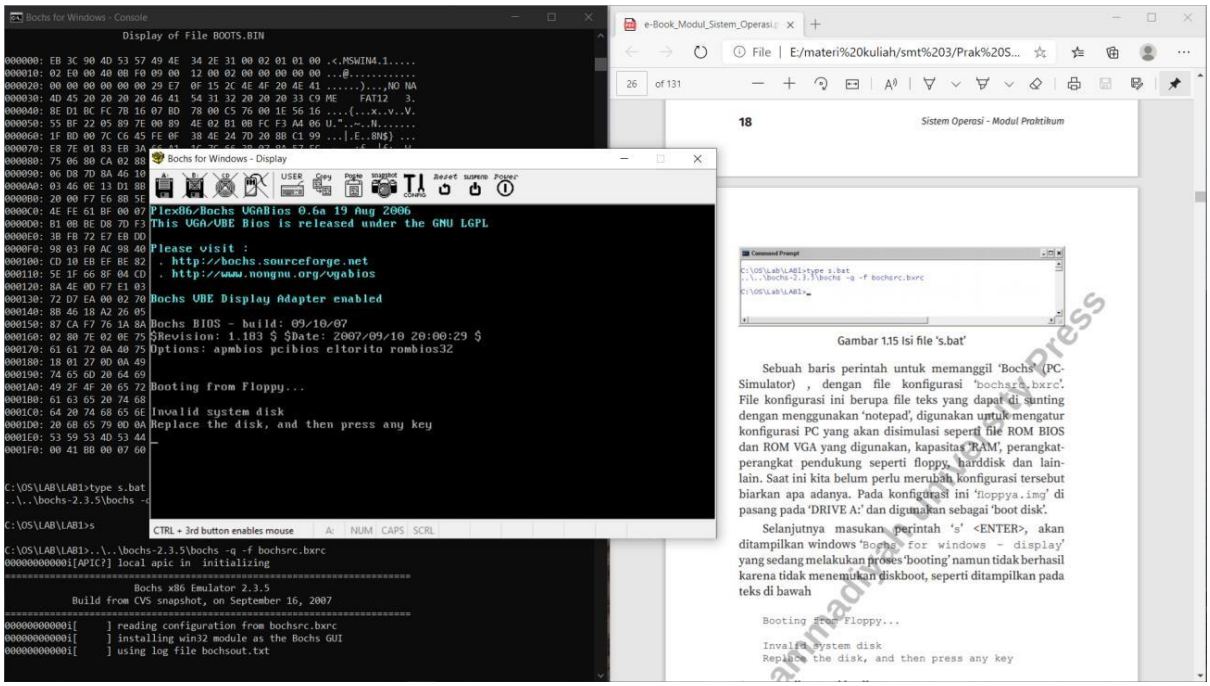
I. Mengetik perintah 'tdump boots. bin'<ENTER>



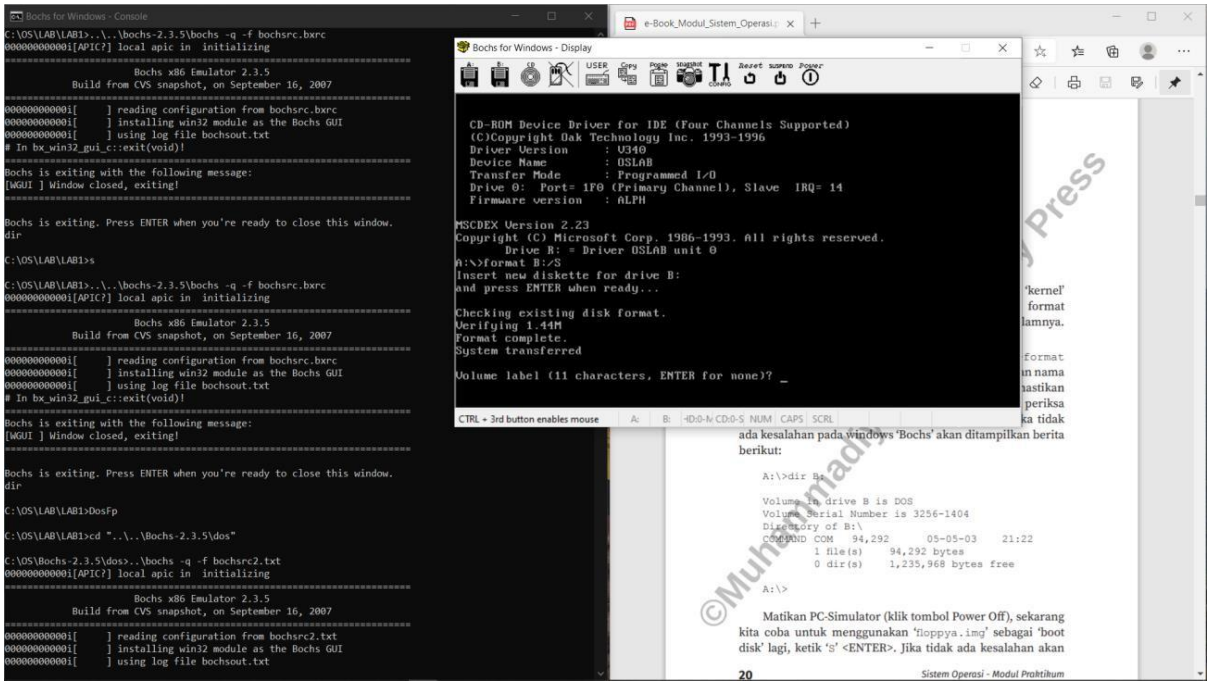
J. Melihat isi file ‘s.bat’ dengan perintah ‘type s.bat’<ENTER>



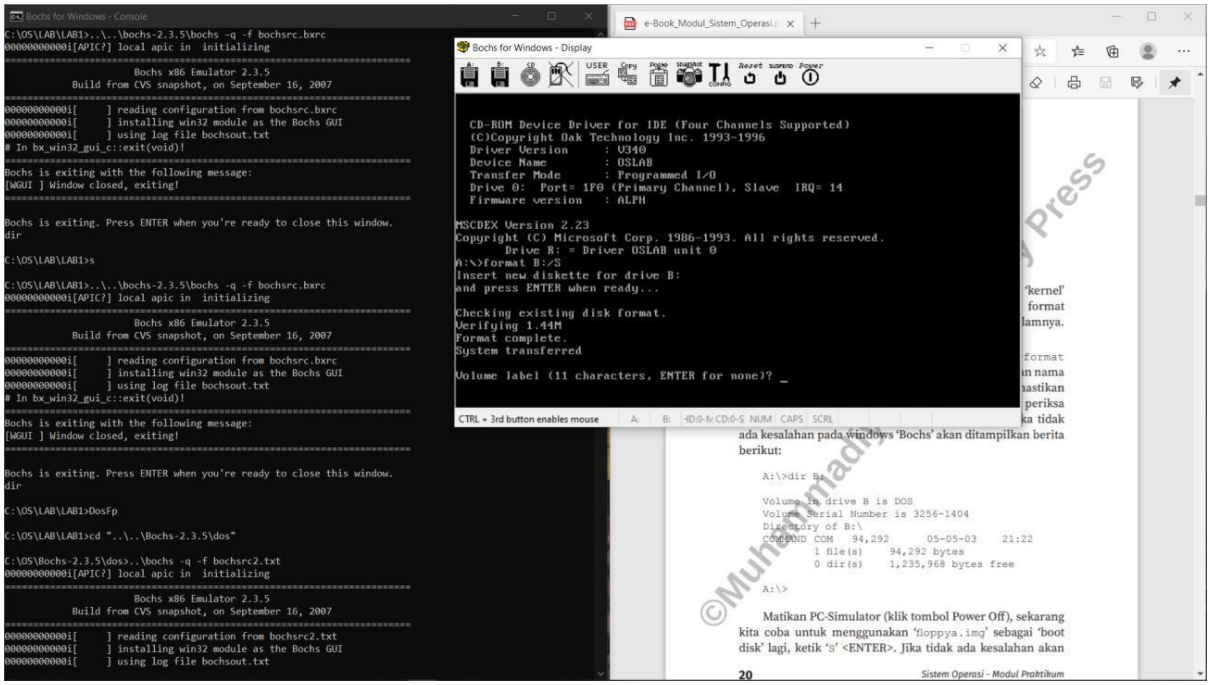
K. Selanjutnya memasukan perintah ‘s’ , akan ditampilkan windows ‘Bochs for windows – display’ yang sedang melakukan proses ‘booting’ namun tidak berhasil karena tidak menemukan diskboot



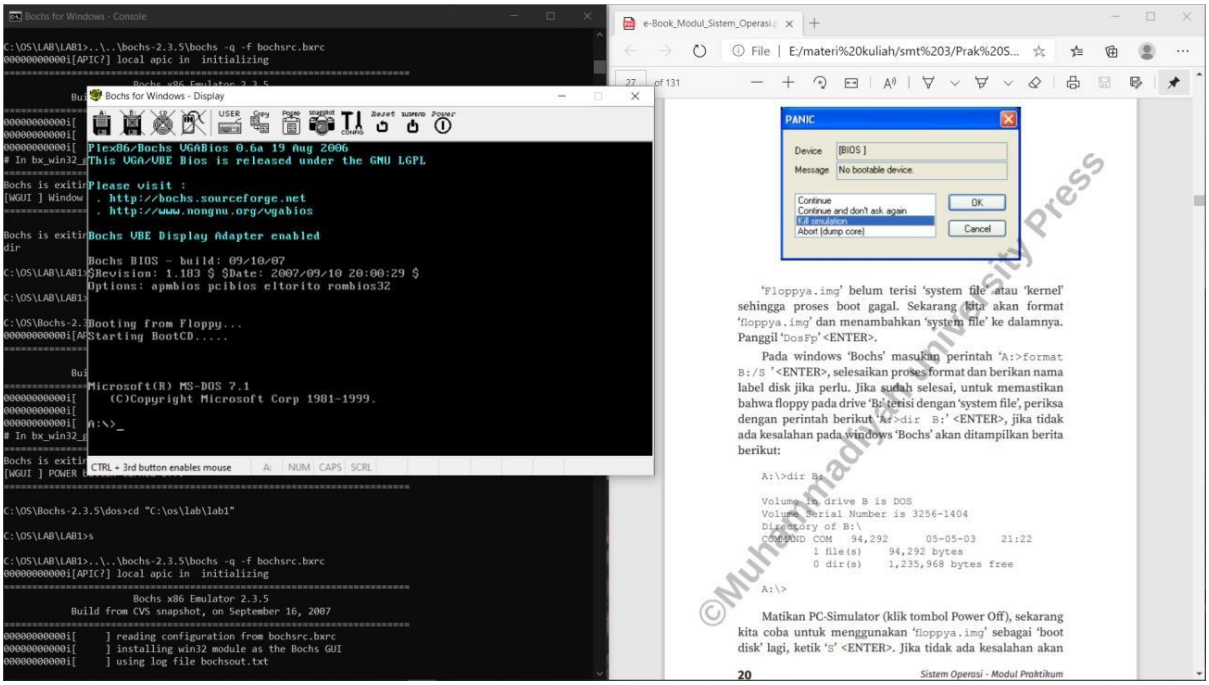
L. Memanggil ‘DosFp’ . Pada windows ‘Bochs’ masukan perintah ‘A:>format B:/S ’ <ENTER>



M. Memanggil 'DosFp' . Pada windows 'Bochs' masukan perintah 'A:>format B:/S ' <ENTER>



N. Mematikan PC-Simulator (klik tombol Power Off), ketik ‘S’<ENTER>



1. **Apa yang dimaksud dengan kode 'ASCII', buatlah tabel kode ASCII lengkap cukup kode ASCII yang standar tidak perlu extended, tuliskan kode ASCII dalam format angka desimal, binary dan hexadesimal serta karakter dan simbol yang dikodekan.**

Kode ASCII (American Standard Codes for International Interchange) adalah kumpulan kode-kode yang dipergunakan untuk mempermudah interaksi antara user dan komputer.

Kode Standar Amerika untuk Pertukaran Informasi atau ASCII (American Standard Code for Information Interchange) merupakan suatu standar internasional dalam kode huruf dan simbol seperti Hex dan Unicode tetapi ASCII lebih bersifat universal, contohnya 124 adalah untuk karakter "|". Ia selalu digunakan oleh komputer dan alat komunikasi lain untuk menunjukkan teks. Kode ASCII sebenarnya memiliki komposisi bilangan biner sebanyak 8 bit. Dimulai dari 00000000 hingga 11111111. Total kombinasi yang dihasilkan sebanyak 256, dimulai dari kode 0 hingga 255 dalam sistem bilangan Desimal.

ASCII Character Set adalah Sebuah standard kode 7 bit yang menggambarkan karakter dari ASCII dengan menggunakan nilai biner. Jangkauan nilai kode ini adalah dari 0-127. Kebanyakan dari Komputer Pribadi (PC) menggunakan perluasan dari kode ASCII berbasis 8 bit, sehingga didapatkan 128 karakter ekstra, yang digunakan sebagai simbol khusus, karakter khusus, dan simbol grafis.

"Interaksi" yang dimunculkan pada artian kalimat tersebut adalah sebuah sarana untuk menyelesaikan permasalahan hubungan antara komputer yang hanya mengenal angka, sedangkan manusia tidak mungkin harus menghafalkan angka yang cukup banyak tersebut dan mempergunakan keyboard sebagai masukan atas perintah yang diinginkannya.

Terdapat dua jenis kode yang berhubungan dengan kode pada keyboard yaitu kode ASCII dan EBCDIC. ASCII adalah kode 7 bit, sehingga karakter digenerate oleh keyboard sebagai 7 bit kode (total jumlahnya ada sebanyak 128 kombinasi yang berbeda). ASCII adalah singkatan dari American Standard Code for Information Interchange. Sedangkan EBCDIC adalah singkatan dari Extended Binary Coded Decimal Interchange Code, dan utamanya digunakan oleh IBM.

Kode ASCII me-representasikan kode-kode untuk :

Angka (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

Huruf (a – z, A – Z)

Simbol (&, ^, %, $ @ ..)

Tombol (Enter, Esc, Backspace, Space, Tab, Shift, Ctrl)

Karakter Grafis (kode ASCII Standar nomor 128 s/d 255)

Kode Komunikasi (ETX, STX, ENQ, ACK ..)

Kompleksnya kode-kode dalam ASCII ini akhirnya perlu untuk dibagi menjadi 2 (dua) bagian, yaitu :

Kode Standard ASCII

Kode Extended ASCII

Masing-masing jenis Kode ASCII tersebut sebanyak 255 buah, atau dapat disebut juga sebagai 255 karakter, karena memang 1 (satu) kode ASCII berukuran 1 Byte (8 bit).

Kode ASCII Standard

Kode ini merepresentasikan angka, huruf serta tombol standar, Enter, Escape, Backspace dan Space. Selain itu juga terdapat karakter-karakter yang tidak terdapat pada keyboard, yang dapat diaktifkan dengan melakukan penekanan tombol kombinasi “Alt” dan angka yang dimaksud, sebagai contoh tombol kombinasi “Alt” dan angka “127″ akan menghasilkan karakter grafis.

Karakter dasar lain juga digunakan untuk komunikasi, seperti yang Anda ketahui bersama, karakter tersebut adalah “ACK” dan “ENQ”. Pada saat akan dilakukan komunikasi pada jaringan dengan protokol Ethernet, maka bentuk komunikasi yang terjadi adalah komputer akan mengirimkan “ACK” (Acknowledge) pada komputer lain yang akan berkomunikasi, jika komputer lain merespon, maka komputer tersebut akan membalasnya dengan mengirim “ENQ” (Enquiry).

Karakter ASCII nomor 5 dan nomor 6 akan bertindak untuk kondisi ini.

Kode ASCII Extended

Kode ASCII Extended akan bertindak sebagai kode perluasan (extended) dari kode ASCII yang ada, karena tidak semuanya mampu tertampung dalam kode ASCII standard.

Kode ASCII jenis ini lebih banyak bertindak sebagai kode-kode tombol khusus, seperti kode untuk tombol F1 s/d F12. Sebagai contoh adalah kode ASCII extended untuk F12 adalah “123″.

Belum lagi dengan tombol kombinasi, misalnya “Alt” dan “F1″, “Ctrl” dan “F1″, atau tombol-tombol yang biasa kita lakukan “Alt” + “F” untuk membuka menu file, “Ctrl” dan “O” untuk

membuka dokumen dsb.

Character Name	Char	Code	Decimal	Binary	Hex
Space			32	00100000	20
Exclamation Point	!	Shift 1	33	00100001	21
Double Quote	"	Shift ‘	34	00100010	22
Pound/Number Sign	#	Shift 3	35	00100011	23
Dollar Sign	$	Shift 4	36	00100100	24
Percent Sign	%	Shift 5	37	00100101	25
Ampersand	&	Shift 7	38	00100110	26
Single Quote	‘	‘	39	00100111	27
Left Parenthesis	(	Shift 9	40	00101000	28
Right Parenthesis	)	Shift 0	41	00101001	29
Asterisk	*	Shift 8	42	00101010	2A
Plus Sign	+	Shift =	43	00101011	2B
Comma	,	,	44	00101100	2C
Hyphen / Minus Sign	-	-	45	00101101	2D
Period	.	.	46	00101110	2E
Forward Slash	/	/	47	00101111	2F
Zero Digit	0	0	48	00110000	30
One Digit	1	1	49	00110001	31
Two Digit	2	2	50	00110010	32
Three Digit	3	3	51	00110011	33
Four Digit	4	4	52	00110100	34

Five Digit	5	5	53	00110101	35
Six Digit	6	6	54	00110110	36
Seven Digit	7	7	55	00110111	37
Eight Digit	8	8	56	00111000	38
Nine Digit	9	9	57	00111001	39
Colon	:	Shift ;	58	00111010	3A
Semicolon	;	;	59	00111011	3B
Less-Than Sign	<	Shift ,	60	00111100	3C
Equals Sign	=	=	61	00111101	3D
Greater-Than Sign	>	Shift .	62	00111110	3E
Question Mark	?	Shift /	63	00111111	3F
At Sign	@	Shift 2	64	01000000	40
Capital A	A	Shift A	65	01000001	41
Capital B	B	Shift B	66	01000010	42
Capital C	C	Shift C	67	01000011	43
Capital D	D	Shift D	68	01000100	44
Capital E	E	Shift E	69	01000101	45
Capital F	F	Shift F	70	01000110	46
Capital G	G	Shift G	71	01000111	47
Capital H	H	Shift H	72	01001000	48
Capital I	I	Shift I	73	01001001	49
Capital J	J	Shift J	74	01001010	4A
Capital K	K	Shift K	75	01001011	4B
Capital L	L	Shift L	76	01001100	4C
Capital M	M	Shift M	77	01001101	4D
Capital N	N	Shift N	78	01001110	4E
Capital O	O	Shift O	79	01001111	4F
Capital P	P	Shift P	80	01010000	50
Capital Q	Q	Shift Q	81	01010001	51
Capital R	R	Shift R	82	01010010	52
Capital S	S	Shift S	83	01010011	53
Capital T	T	Shift T	84	01010100	54
Capital U	U	Shift U	85	01010101	55
Capital V	V	Shift V	86	01010110	56
Capital W	W	Shift W	87	01010111	57
Capital X	X	Shift X	88	01011000	58
Capital Y	Y	Shift Y	89	01011001	59
Capital Z	Z	Shift Z	90	01011010	5A
Left Bracket	[	[	91	01011011	5B
Backward Slash	\	\	92	01011100	5C
Right Bracket	]	]	93	01011101	5D
Caret	^	Shift 6	94	01011110	5E
Underscore	_	Shift -	95	01011111	5F
Back Quote	`	`	96	01100000	60
Lower-case A	a	A	97	01100001	61
Lower-case B	b	B	98	01100010	62

Lower-case C	c	C	99	01100011	63
Lower-case D	d	D	100	01100100	64
Lower-case E	e	E	101	01100101	65
Lower-case F	f	F	102	01100110	66
Lower-case G	g	G	103	01100111	67
Lower-case H	h	H	104	01101000	68
Lower-case I	I	I	105	01101001	69
Lower-case J	j	J	106	01101010	6A
Lower-case K	k	K	107	01101011	6B
Lower-case L	l	L	108	01101100	6C
Lower-case M	m	M	109	01101101	6D
Lower-case N	n	N	110	01101110	6E
Lower-case O	o	O	111	01101111	6F
Lower-case P	p	P	112	01110000	70
Lower-case Q	q	Q	113	01110001	71
Lower-case R	r	R	114	01110010	72
Lower-case S	s	S	115	01110011	73
Lower-case T	t	T	116	01110100	74
Lower-case U	u	U	117	01110101	75
Lower-case V	v	V	118	01110110	76
Lower-case W	w	W	119	01110111	77
Lower-case X	x	X	120	01111000	78
Lower-case Y	y	Y	121	01111001	79
Lower-case Z	z	Z	122	01111010	7A
Left Brace	{	Shift [	123	01111011	7B
Vertical Bar	|	Shift \	124	01111100	7C
Right Brace	}	Shift]	125	01111101	7D

2. Carilah daftar perintah bahasa assembly untuk mesin intel keluarga x86 lengkap (dari buku referensi atau internet). Daftar perintah ini dapat digunakan sebagai pedoman untuk memahami program ‘boot.asm’ dan ‘kernel.asm’

Terbagi menjadi 3 bagian utama yaitu :

1. Komentar

Komentar diawali dengan tanda titik koma (;).

; ini adalah komentar 2.

Label

Label diakhiri dengan tanda titik dua (:). Contoh: main:

,loop: ,proses: ,keluar:

3. Assembler directives

Directives adalah perintah yang ditujukan kepada assembler ketika sedang menerjemahkan program kita ke bahasa mesin.

Directive dimulai dengan tanda titik. **.model** : memberitahu assembler berapa memori yang akan dipakai oleh program kita.

Ada model tiny, model small, model compact, model medium, model large, dan model huge**.**

.data : memberitahu assembler bahwa bagian di bawah ini adalah data program.

.code : memberitahu assembler bahwa bagian di bawah ini adalah instruksi program.

.stack : memberitahu assembler bahwa program kita memiliki stack.

Program EXE harus punya stack. Kira-kira yang penting itu dulu.

Semua directive yang dikenal assembler
adalah: .186 .286 .286c .286p .287 .386 .386c .386p .387 .486 .486p
.8086 .8087

.alpha .break .code .const .continue .cref .data .data? .dosseg .else .el seif .endif .endw .err .err1 .err2 .errb

.errdef .errdif .errdifi .erre .erridn .erridni .errnb .errndef .errnz .exit .f ardata .fardata? .if .lall .lfcond .list .listall .listif .listmacro

.listmacroall .model .no87 .nocref .nolist .nolistif .nolistmacro .radix .r epeat .sall .seq .sfcond .stack

.startup .tfcond .type .until .untilcxz .while .xall .xcref .xlist.

Definisi data

DB : define bytes. Membentuk data byte demi byte. Data bisa data numerik maupun teks.

catatan: untuk membentuk data string, pada akhir string harus diakhiri tanda dolar ($).

sintaks: {label} DB {data} contoh: teks1 db "Hello world $" **DW** :
define words.

Membentuk data word demi word (1 word = 2 byte).

sintaks: {label} DW {data} contoh: kucing dw ?, ?, ? ;mendefinisikan tiga slot 16-bit yang isinya don't care

(disimbolkan dengan tanda tanya)

DD : define double words. Membentuk data doubleword demi doubleword (4 byte).

sintaks: {label} DD {data} **EQU** : equals. Membentuk konstanta.
sintaks: {label} EQU {data}

contoh: sepuluh EQU 10

Ada assembly yang melibatkan bilangan pecahan (floating point), bilangan bulat (integer),
DF (define far words),
DQ (define quad words), dan DT (define ten bytes).

Perpindahan data

MOV : move. Memindahkan suatu nilai dari register ke memori, memori ke register, atau register ke register.

sintaks: MOV {tujuan}, {sumber}

contoh:

mov AX, 4C00h ;mengisi register AX dengan 4C00(hex).

mov BX, AX ;menyalin isi AX ke BX. mov CL, [BX] ;mengisi register CL dengan data di memori yang alamatnya ditunjuk BX.

mov CL, [BX] + 2 ;mengisi CL dengan data di memori yang alamatnya ditunjuk BX lalu geser maju 2 byte.

mov [BX], AX ;menyimpan nilai AX pada tempat di memori yang ditunjuk BX. mov [BX] – 1, 00101110b

;menyimpan 00101110(bin) pada alamat yang ditunjuk BX lalu geser mundur 1 byte.

LEA : load effective address. Mengisi suatu register dengan alamat offset sebuah data.

sintaks: LEA {register}, {sumber} contoh: lea DX, teks1 **XCHG** :
exchange. Menukar dua buah register langsung.

sintaks: XCHG {register 1}, {register 2} Kedua register harus punya ukuran yang sama.

Bila sama-sama 8 bit (misalnya AH dengan BL) atau sama-sama 16 bit (misalnya CX dan DX),

maka pertukaran bisa dilakukan. Sebenarnya masih banyak perintah perpindahan data,

misalnya IN, OUT, LODS, LODSB, LODSW, MOVS, MOVSB, MOVSW, LDS, LES, LAHF, SAHF, dan XLAT.

Operasi logika

AND : melakukan bitwise and. sintaks: AND {register}, {angka} AND {register 1}, {register 2} hasil disimpan di register 1.

contoh: mov AL, 00001011b mov AH, 11001000b and AL, AH ;sekarang AL
berisi 00001000(bin),

sedangkan AH tidak berubah.

OR : melakukan bitwise or. sintaks: OR {register}, {angka} OR {register 1}, {register 2} hasil disimpan di register 1.

NOT : melakukan bitwise not (*one's complement*) sintaks: NOT {register} hasil

disimpan di register itu sendiri.

XOR : melakukan bitwise eksklusif or. sintaks: XOR {register}, {angka} XOR {register 1}, {register 2} hasil disimpan di register 1. Tips: sebuah register yang di-XOR-kan dengan dirinya sendiri akan menjadi berisi nol.

SHL : shift left. Menggeser bit ke kiri. Bit paling kanan diisi nol.
sintaks: SHL {register}, {banyaknya}

SHR : shift right. Menggeser bit ke kanan. Bit paling kiri diisi nol.
sintaks: SHR {register}, {banyaknya}

ROL : rotate left. Memutar bit ke kiri. Bit paling kiri jadi paling kanan kali ini. sintaks: ROL {register},

{banyaknya} Bila banyaknya rotasi tidak disebutkan, maka nilai yang ada di CL akan digunakan sebagai banyaknya rotasi.

ROR : rotate right. Memutar bit ke kanan. Bit paling kanan jadi paling kiri. sintaks: ROR {register},

{banyaknya} Bila banyaknya rotasi tidak disebutkan, maka nilai yang ada di CL akan digunakan sebagai banyaknya rotasi.

Ada lagi : RCL dan RCR.

Operasi matematika

ADD : add. Menjumlahkan dua buah register.

sintaks: ADD {tujuan}, {sumber} operasi yang terjadi: tujuan = tujuan + sumber.

carry (bila ada) disimpan di CF.

ADC : add with carry. Menjumlahkan dua register dan carry flag (CF).

sintaks: ADC {tujuan}, {sumber} operasi yang terjadi: tujuan = tujuan + sumber + CF.

carry (bila ada lagi) disimpan lagi di CF.

INC : increment. Menjumlah isi sebuah register dengan 1.

Bedanya dengan ADD, perintah INC hanya memakan 1 byte memori sedangkan ADD pakai 3 byte.

sintaks: INC {register}

SUB : substract. Mengurangkan dua buah register.

sintaks: SUB {tujuan}. {sumber} operasi yang terjadi: tujuan = tujuan – sumber.

borrow (bila terjadi) menyebabkan CF bernilai 1.

SBB : substract with borrow. Mengurangkan dua register dan carry flag (CF).
sintaks: SBB {tujuan}, {sumber} operasi yang terjadi: tujuan = tujuan – sumber – CF.
borrow (bila terjadi lagi) menyebabkan CF dan SF (sign flag) bernilai 1.

DEC : decrement. Mengurang isi sebuah register dengan 1.

Jika SUB memakai 3 byte memori, DEC hanya memakai 1 byte. sintaks: DEC
{register}

MUL : multiply. Mengalikan register dengan AX atau AH.

sintaks: MUL {sumber} Bila register sumber adalah 8 bit,

maka isi register itu dikali dengan isi AL, kemudian disimpan di AX.

Bila register sumber adalah 16 bit, maka isi register itu dikali dengan isi AX,

kemudian hasilnya disimpan di DX:AX. Maksudnya, DX berisi high order byte-nya, AX berisi low order byte-nya.

IMUL : signed multiply. Sama dengan MUL,

hanya saja IMUL menganggap bit-bit yang ada di register sumber sudah dalam bentuk *two's complement*.
sintaks: IMUL {sumber}

DIV : divide. Membagi AX atau DX:AX dengan sebuah register.

sintaks: DIV {sumber} Bila register sumber adalah 8 bit (misalnya:
BL), maka operasi yang terjadi: -AX dibagi BL,

-hasil bagi disimpan di AL, -sisa bagi disimpan di AH.

Bila register sumber adalah 16 bit (misalnya: CX), maka operasi yang
terjadi: -DX:AX dibagi CX, -hasil bagi disimpan di AX, -sisa bagi disimpan di DX.

IDIV : signed divide. Sama dengan DIV, hanya saja IDIV menganggap bit-bit yang ada di register sumber sudah dalam bentuk *two's complement*.

sintaks: IDIV {sumber}

NEG : negate. Membuat isi register menjadi negatif (*two*'*s complement*).
Bila mau *one*'*s complement*, gunakan perintah NOT. sintaks: NEG {register} hasil disimpan di register itu sendiri.

Pengulangan

LOOP : loop. Mengulang sebuah proses. Pertama register CX dikurangi satu.
Bila CX sama dengan nol, maka looping berhenti. Bila tidak nol, maka lompat ke label tujuan.
sintaks: LOOP {label tujuan} Tips: isi CX dengan nol untuk mendapat jumlah pengulangan terbanyak.
Karena nol dikurang satu sama dengan -1, atau dalam notasi *two*'*s complement* menjadi FFFF(hex) yang sama dengan 65535(dec).
LOOPE : loop while equal. Melakukan pengulangan selama CX ≠ 0 dan ZF = 1. CX tetap dikurangi 1 sebelum diperiksa.
sintaks: LOOP {label tujuan}

LOOPZ : loop while zero. Identik dengan LOOPE.

LOOPNE : loop while not equal.

Melakukan pengulangan selama CX ≠ 0 dan ZF = 0. CX tetap dikurangi 1 sebelum diperiksa.
sintaks: LOOPNE {label tujuan}

LOOPNZ : loop while not zero. Identik dengan LOOPNE.

REP : repeat. Mengulang perintah sebanyak CX kali. sintaks: REP {perintah assembly}
contoh:
mov CX, 05 rep inc BX ;register BX ditambah 1 sebanyak 5x.

REPE : repeat while equal. Mengulang perintah sebanyak CX kali, tetapi pengulangan segera dihentikan bila didapati ZF = 1.

sintaks: REPE {perintah assembly}

REPZ : repeat while zero. Identik dengan REPE.

REPNE : repeat while not equal. Mengulang perintah sebanyak CX kali, tetapi pengulangan segera dihentikan bila didapati ZF = 0.

sintaks: REPNE {perintah assembly}

REPNZ : repeat while not zero. Identik dengan REPNE.

Perbandingan

CMP : compare. Membandingkan dua buah operand. Hasilnya mempengaruhi sejumlah flag register.

sintaks: CMP {operand 1}, {operand 2}. Operand ini bisa register dengan register , register dengan isi memori, atau register dengan angka.

CMP tidak bisa membandingkan isi memori dengan isi memori.
Hasilnya adalah:

Kasus	Bila operand 1 < operand 2	Bila operand 1 = operand 2	Bila operand 1 > operand 2
Signed binary	OF=1,SF= 1,ZF=0	OF=0,SF= 0,ZF=1	OF=0,SF= 0,ZF=0
Unsigned binary	CF=1,ZF=0	CF=0,ZF=1	CF=0,ZF=0

Lompat-lompat

JMP: jump. Lompat tanpa syarat. Lompat begitu saja. sintaks: JMP {label tujuan}

Lompat bersyarat sintaksnya sama dengan JMP, yaitu perintah jump diikuti label tujuan.

PERINTAH	ARTI	SYARAT	KASUS	KETERANGAN ("OP" = OPERAND)	MENGIKUTI CMP?
JA	jump if above				
JNBE	jump if not below or equal	CF=0 ZF=0	unsigned	lompat bila op 1 > op 2	ya
JB	jump if below				
JNAE	jump if not above or equal	CF=1 ZF=0	unsigned	lompat bila op 1 < op 2	ya
JAE	jump if above or equal				
JNB	jump if not below	CF=0 ZF=1	unsigned	lompat bila op 1 ≥ op 2	ya
JBE	jump if below or equal				
JNA	jump if not above	CF=1 ZF=1	unsigned	lompat bila op 1 ≤ op 2	ya
JG	jump if greater				
JNLE	jump if not less or equal	OF=0 ZF=0	signed	lompat bila op 1 > op 2	ya
JGE	jump if greater or equal				
JNL	jump if not less than	OF=0 ZF=1	signed	lompat bila op 1 ≥ op 2	ya
JL	jump if less than				
JNGE	jump if not greater or equal	OF=1 ZF=0	signed	lompat bila op 1 < op 2	ya
JLE	jump if less or equal				
JNG	jump if not greater	OF=1 ZF=1	signed	lompat bila op 1 ≤ op 2	ya
JE	jump if equal	ZF=1	kedua nya	lompat bila op 1 = op 2	ya
JZ	jump if zero	ZF=1	kedua nya	lompat bila op 1 = op 2	ya
JNE	jump if not equal	ZF=0	kedua nya	lompat bila op 1 ≠ op 2	ya
JNZ	jump if not zero	ZF=0	kedua nya	lompat bila op 1 ≠ op 2	ya
JC	jump if carry	CF=1	N/A	lompat bila carry flag = 1	tidak
JNC	jump if not carry	CF=0	N/A	lompat bila carry flag = 0	tidak
JP	jump on parity			lompat bila parity flag = 1	
	jump on			lompat bila	tidak

JPE	parity even	PF=1	N/A	bilangan genap	selalu
JNP	jump on not parity	PF=0	N/A	lompat bila parity flag = 0	tidak selalu
JPO	jump on parity odd			lompat bila bilangan ganjil	
JO	jump if overflow	OF=1	N/A	lompat bila overflow flag = 1	tidak
JNO	jump if not overflow	OF=0	N/A	lompat bila overflow flag = 0	tidak
JS	jump if sign	SF=1	N/A	lompat bila bilangan negatif	tidak
JCXZ	jump if CX is zero	CX = 0000	N/A	lompat bila CX berisi nol	tidak

Operasi stack

PUSH : push. Menambahkan sesuatu ke stack.

Sesuatu ini harus register berukuran 16 bit (pada 386+ harus 32 bit), tidak boleh angka, tidak boleh alamat memori.
Maka Anda tidak bisa mem-push register 8-bit seperti AH, AL, BH, BL, dan kawan-kawannya.
sintaks: push {register 16-bit sumber}

contoh: push DX push AX Setelah operasi push, register SP (stack pointer) otomatis dikurangi 2 (karena datanya 2 byte).
Makanya, “top” dari stack seakan-akan “tumbuh turun”.

POP : pop. Mengambil sesuatu dari stack.

Sesuatu ini akan disimpan di register tujuan dan harus 16-bit. Maka Anda tidak bisa mem-pop menuju AH, AL, dkk.
sintaks: POP {register 16-bit tujuan}

contoh: POP BX Setelah operasi pop, register SP otomatis ditambah 2 (karena 2 byte), sehingga “top” dari stack “naik” lagi.

Tip: karena register segmen tidak bisa diisi langsung nilainya, Anda bisa menggunakan stack sebagai perantaranya.
Contoh kodenya: mov AX, seg teks1 push AX pop DS

PUSHF : push flags. Mem-push **semua** isi register flag ke dalam stack.

Biasa dipakai untuk mem*backup* data di register flag sebelum operasi matematika.
Sintaks: PUSHF ;(saja).

POPF : pop flags. Lawan dari pushf. Sintaks: POPF ;(saja).

POPA : pop all general-purpose registers.

Adalah ringkasan dari sejumlah perintah dengan urutan: *pop DI pop SI pop*

BP pop SP pop BX pop DX pop CX pop AX Urutan sudah ditetapkan seperti

itu.

sintaks: POPA ;(saja). Jauh lebih cepat mengetikkan POPA daripada mengetik POP-POP-POP yang banyak itu.

PUSHA : push all general-purpose registers. Lawan dari POPA,

dimana PUSHA adalah singkatan dari sejumlah perintah dengan urutan yang sudah ditetapkan:

push AX push CX push DX push BX push SP push BP push SI push DI

Operasi pada register flag

CLC : clear carry flag. Menjadikan CF = 0. Sintaks: CLC ;(saja).

STC : set carry flag. Menjadikan CF = 1. Sintaks: STC ;(saja).

CMC : complement carry flag. Melakukan operasi NOT pada CF. Yang tadinya 0 menjadi 1, dan sebaliknya.

CLD : clear direction flag. Menjadikan DF = 0. Sintaks: CLD ;(saja).

STD : set direction flag. Menjadikan DF = 1.

CLI : clear interrupt flag. Menjadikan IF = 0, sehingga interrupt ke CPU akan di-disable.

Biasanya perintah CLI diberikan sebelum menjalankan sebuah proses penting yang riskan gagal bila diganggu.

STI : set interrupt flag. Menjadikan IF = 1.

Perintah lainnya :

ORG : origin. Mengatur awal dari program (bagian static data).

Analoginya seperti mengatur dimana letak titik (0, 0) pada koordinat Cartesius.
sintaks: ORG {alamat awal}

Pada program COM (program yang berekstensi .com), harus ditulis “ORG 100h” untuk mengatur alamat mulai dari progam pada 0100(hex),

karena dari alamat 0000(hex) sampai 00FF(hex) sudah dipesan oleh sistem operasi (DOS).
INT : interrupt. Menginterupsi prosesor.

Prosesor akan:

1. Membackup data registernya saat itu,
2. Menghentikan apa yang sedang dikerjakannya,
3. Melompat ke bagian interrupt-handler (entah dimana kita tidak tahu, sudah ditentukan BIOS dan DOS),
4. Melakukan interupsi,
5. Mengembalikan data registernya,
6. Meneruskan pekerjaan yang tadi ditunda.

sintaks: INT {nomor interupsi}

IRET : interrupt-handler return.

Kita bisa membuat interrupt-handler sendiri dengan berbagai cara.

Perintah IRET adalah perintah yang menandakan bahwa interrupt-handler kita selesai,
dan prosesor boleh melanjutkan pekerjaan yang tadi tertunda. **CALL** : call

procedure. Memanggil sebuah prosedur. sintaks: CALL {label nama prosedur}

RET : return. Tanda selesai prosedur.

Setiap prosedur harus memiliki RET di ujungnya.

sintaks: RET ;(saja)

HLT : halt. Membuat prosesor menjadi tidak aktif.

Prosesor harus mendapat interupsi dari luar atau di-reset supaya aktif kembali.

Jadi, jangan gunakan perintah HLT untuk mengakhiri program!!

Sintaks: HLT ;(saja). **NOP** : no operation.

Perintah ini memakan 1 byte di memori tetapi tidak menyuruh prosesor melakukan apa-apa selama 3 clock prosesor.

Berikut contoh potongan program untuk melakukan *delay* selama 0,1 detik pada prosesor Intel 80386 yang berkecepatan 16 MHz.

mov ECX, 533333334d ;ini adalah bilangan desimal idle: nop loop idle