TUGAS MODUL 1

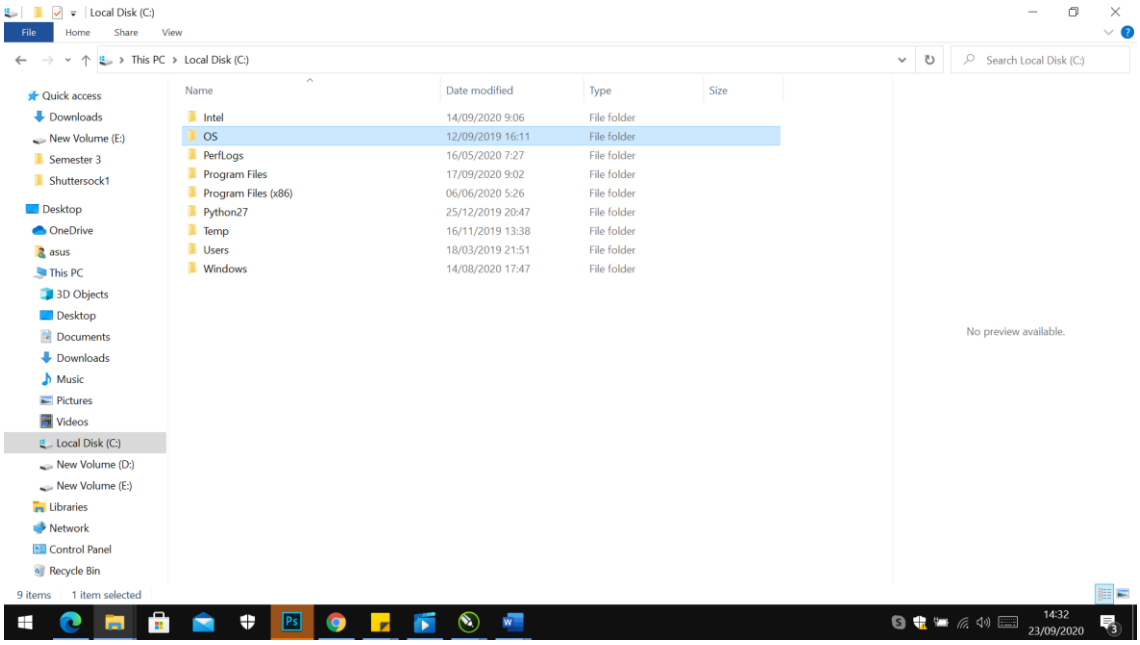
Nama : Sherly Maycana Puspita Anwar

NIM : L200190211

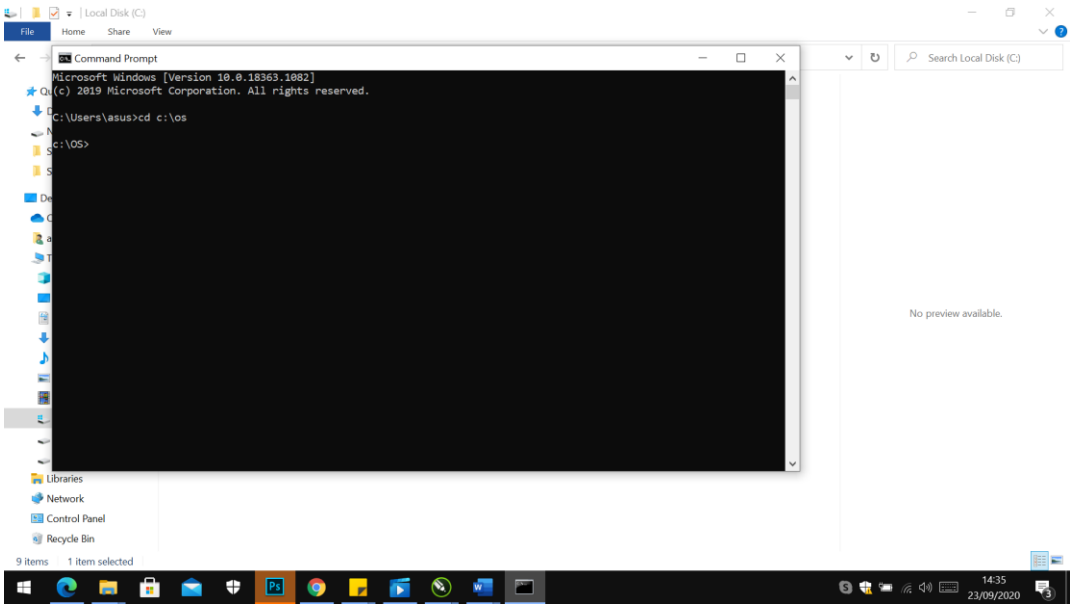
Mata kuliah : Praktikum Sistem Operasi

Langkah – Langkah Modul 1

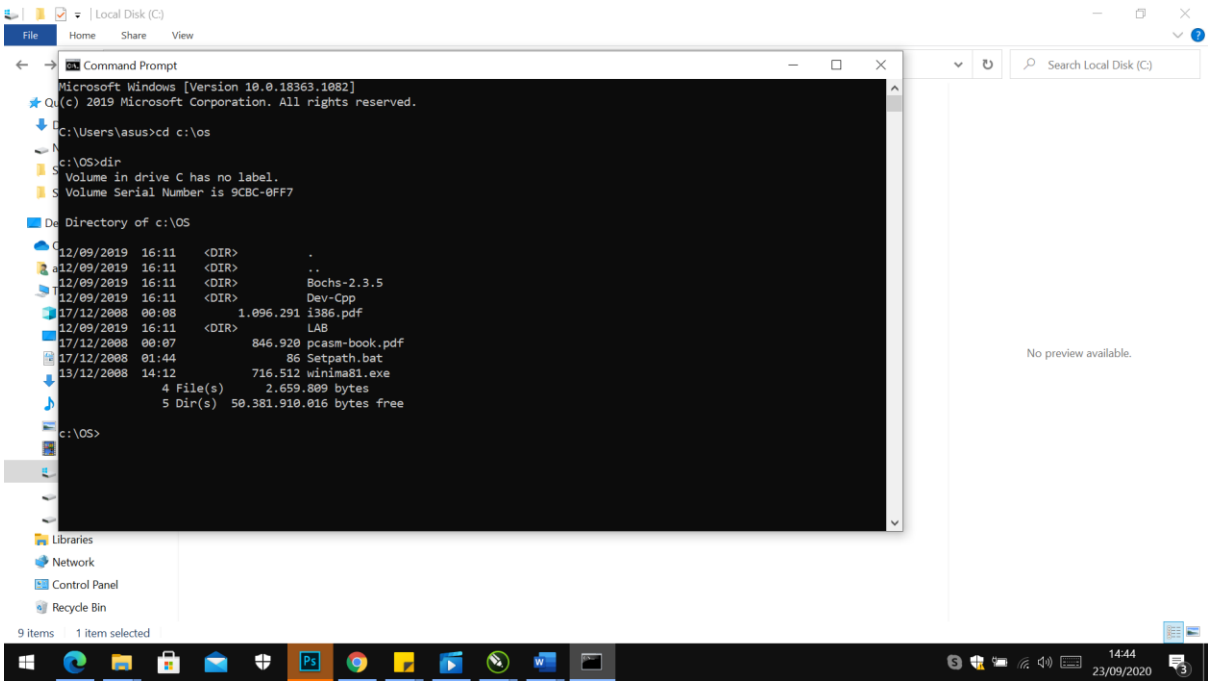
1. Extract file SO.rar kemudian copypaste pada directory C



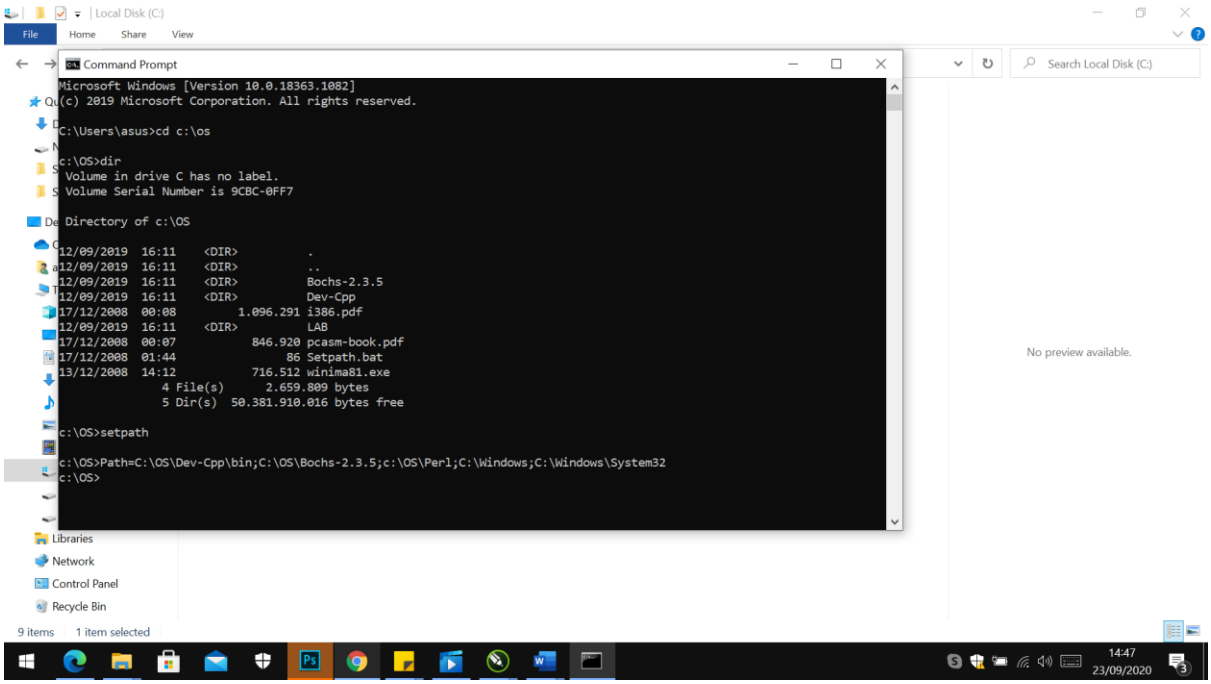
2. Buka Command Prompt, ketikkan perintah " cd c:\os " untuk masuk / berpindah direktori



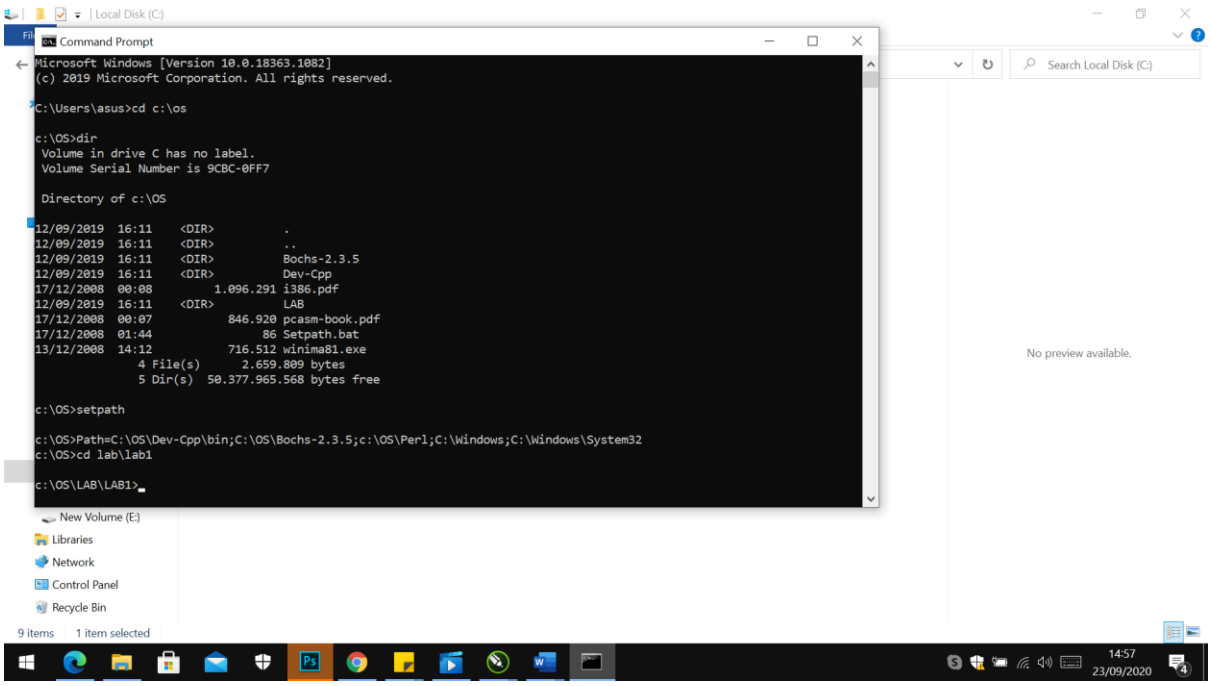
3. Ketikkan perintah “ dir ”, untuk melihat isi direktori dalam folder tersebut.



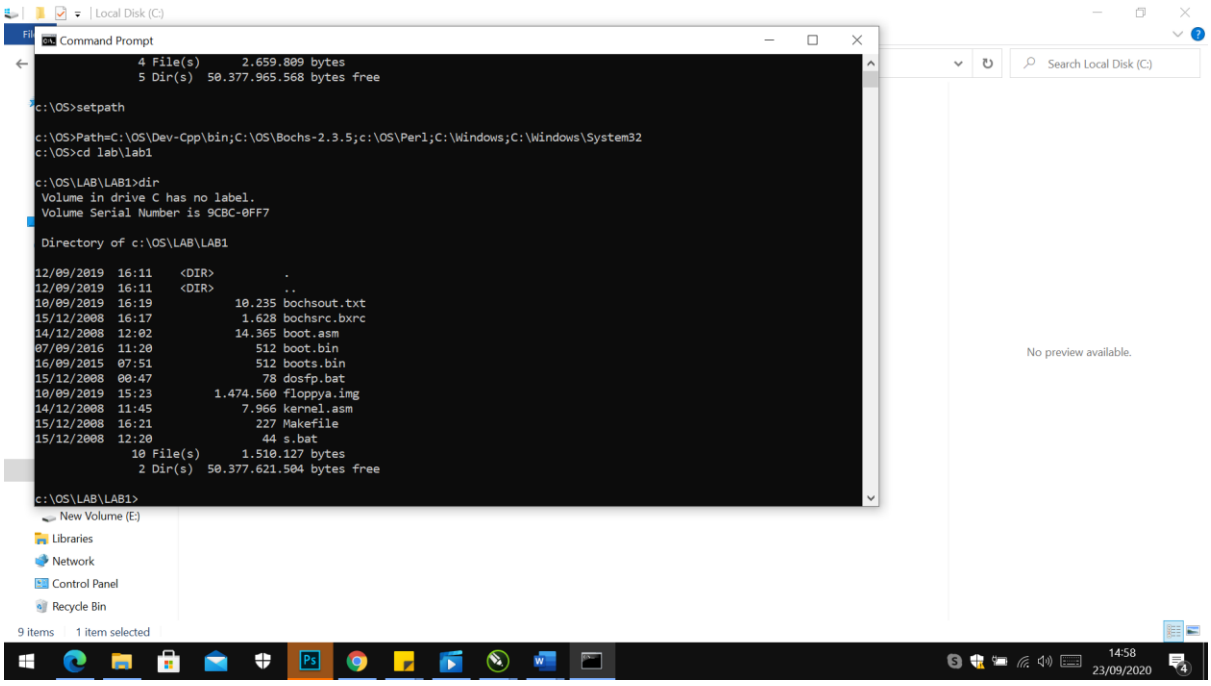
4. Jalankan file “ setpath ”, untuk menjalankannya ketik “ setpath ”, kemudian Enter.



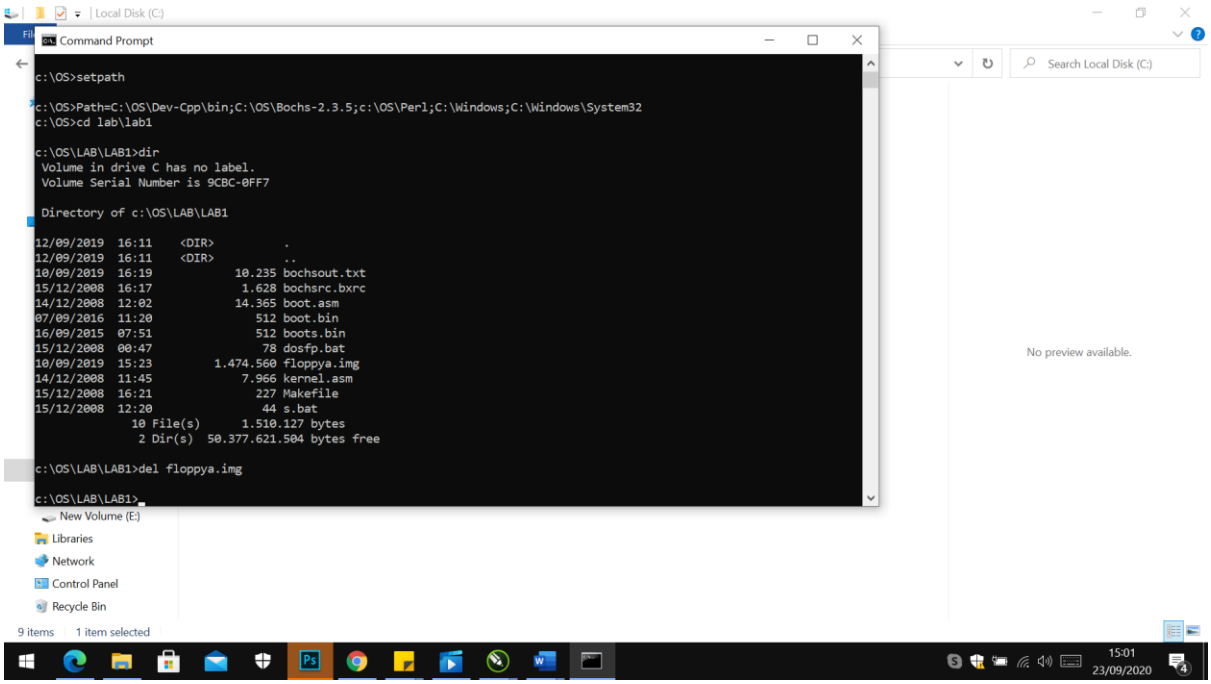
5. Masuk ke dalam direktori kerja LAB1 dengan mengetikkan “ cd lab\lab1”



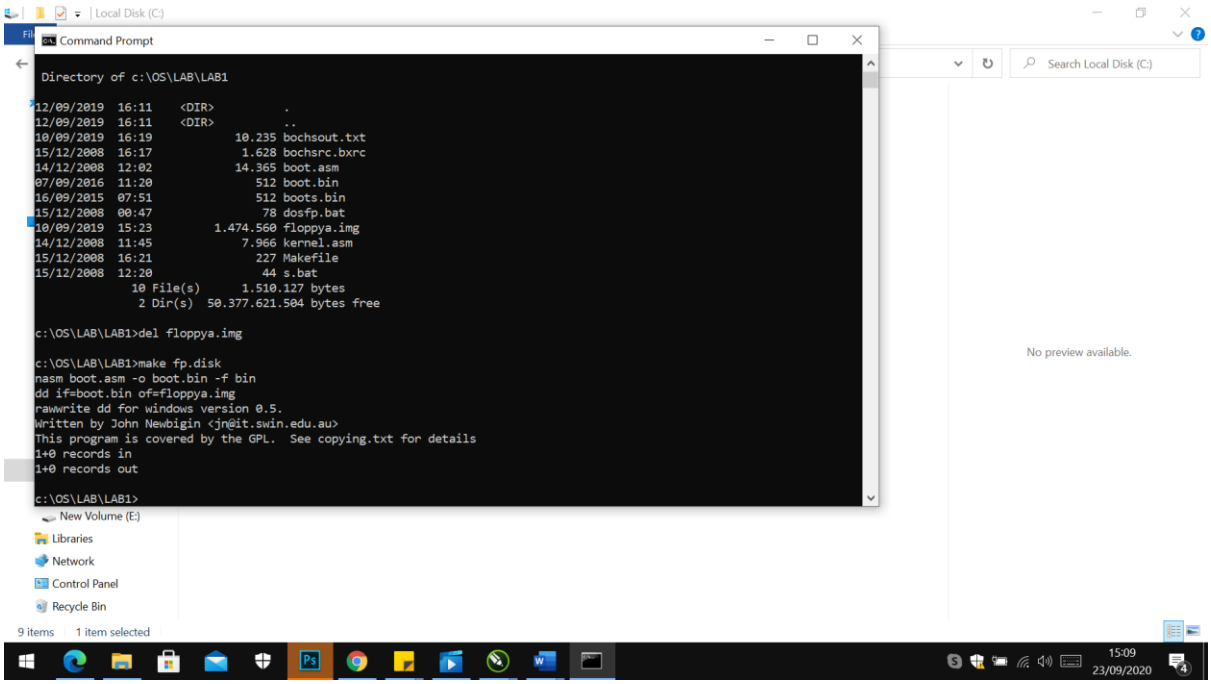
6. Kemudian ketikkan perintah “ dir “ untuk melihat isi direktori dari folder LAB1



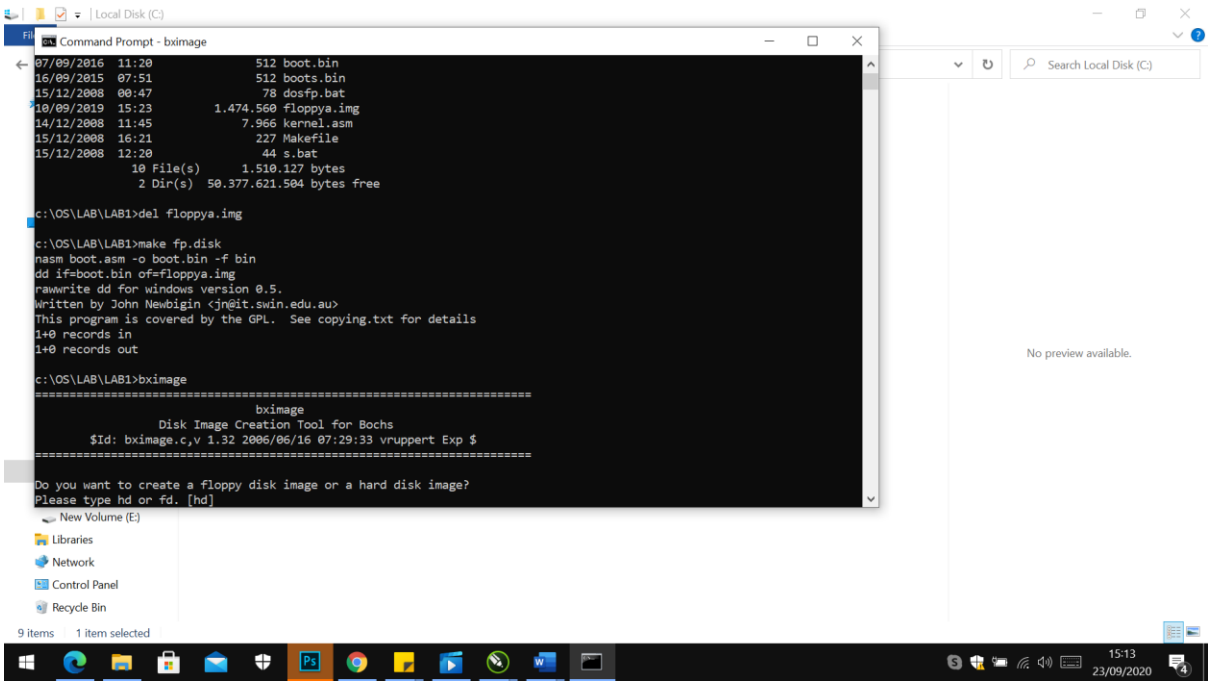
7. Hapus file floppya.img yang sudah ada pada direktori kerja dengan mengetikkan “ del floppya.img ”



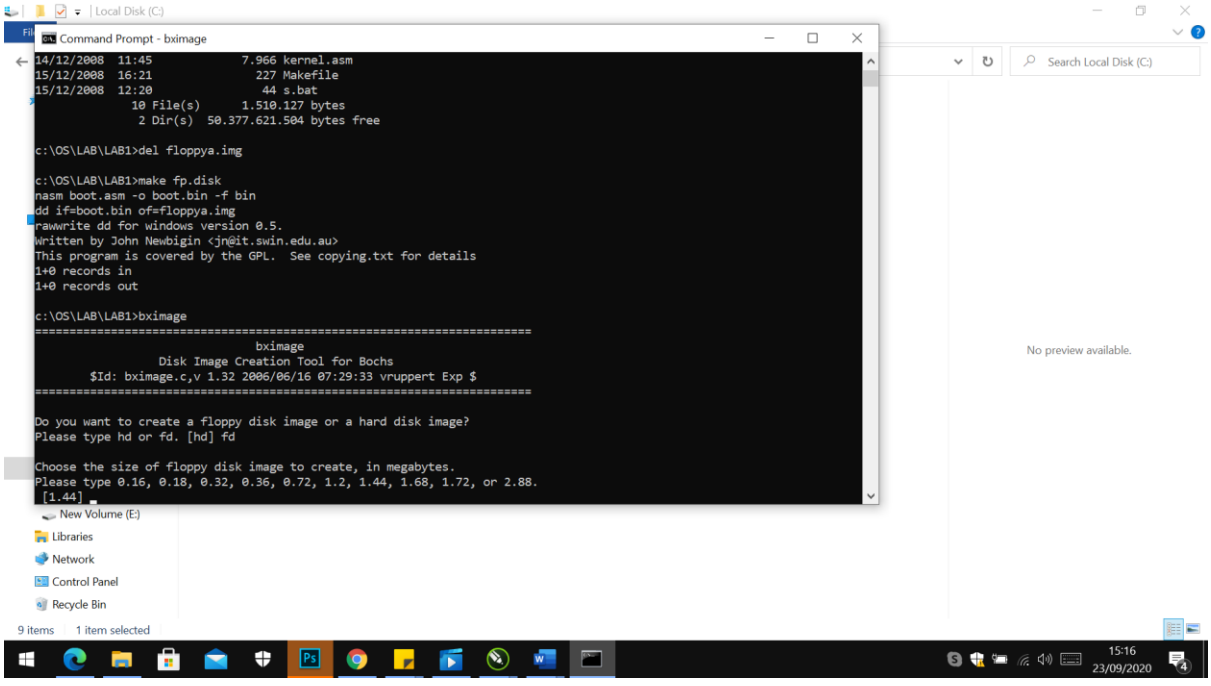
8. Ketikkan “ make fp.disk ” untuk melakukan kompilasi terhadap source code program ‘boot.asm’, sebagai outputnya file ‘boot.bin’ dan isinya disalin ke dalam bootsector file image floppy ‘floppya.img’



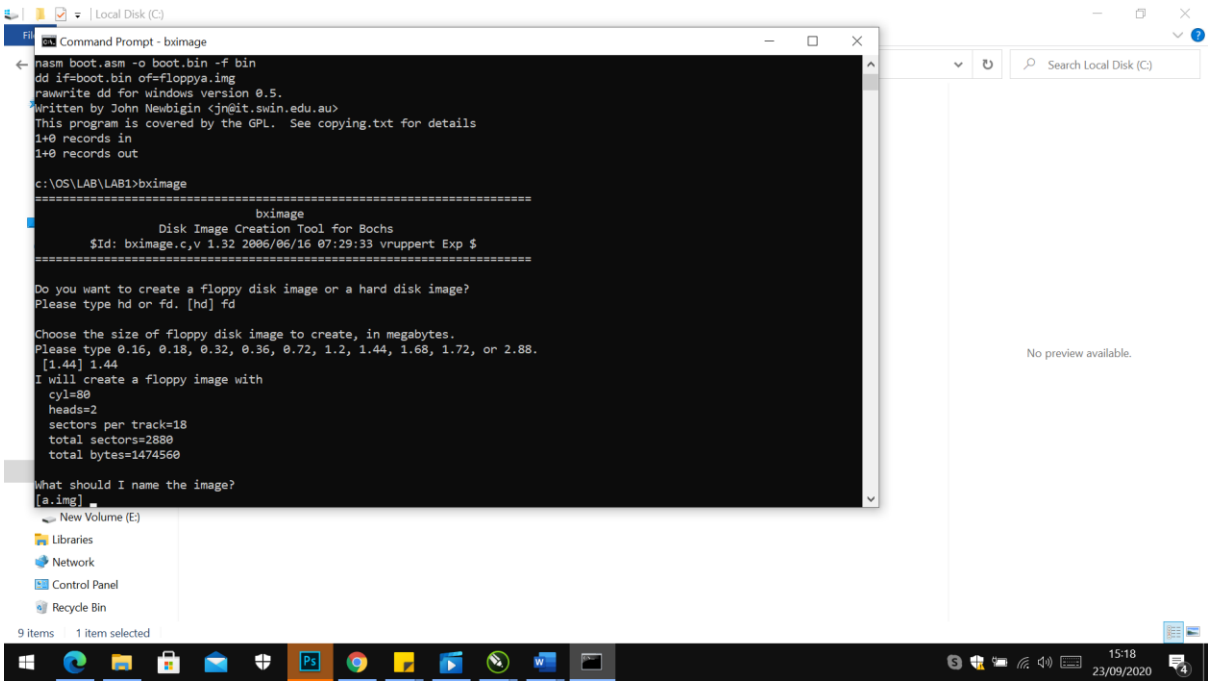
9. Perintah “ bximage ” digunakan untuk membuat file image floppy baru



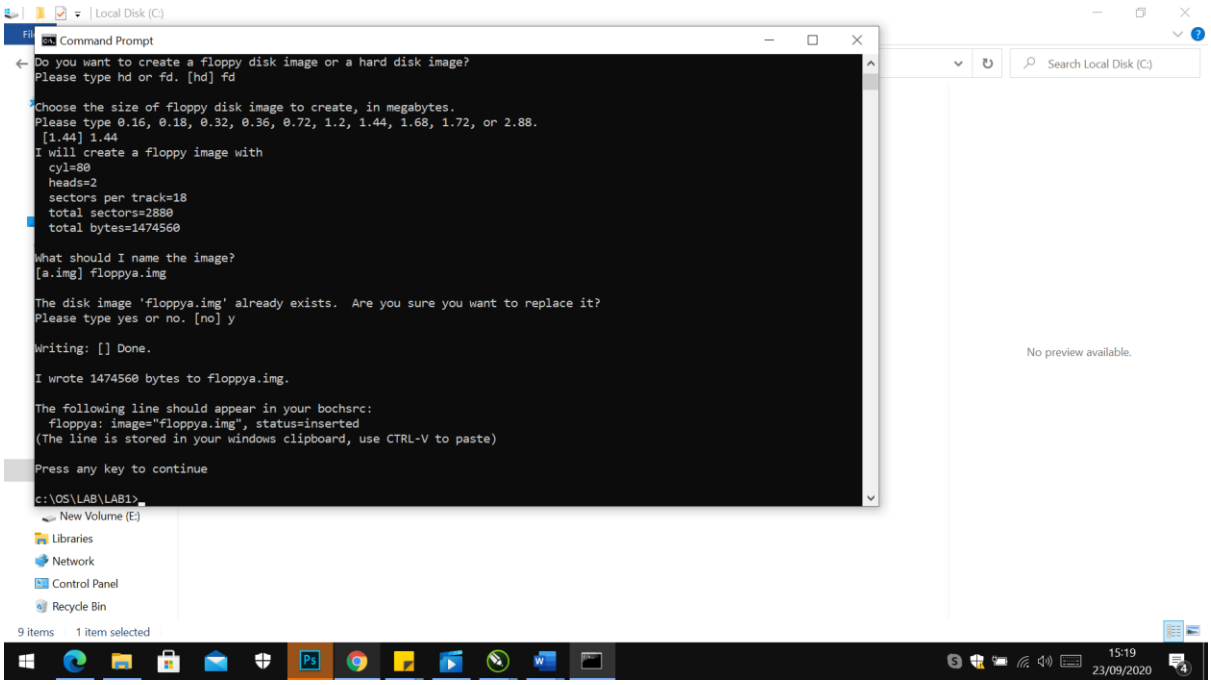
10. Terdapat 2 pilihan, yaitu untuk membuat file fd atau hd. Kita pilih fd karena ukuran filenya lebih kecil. Ketik “ fd ”, lalu Enter.



11. Pilih type kapasitas 1.44 MB, Enter.

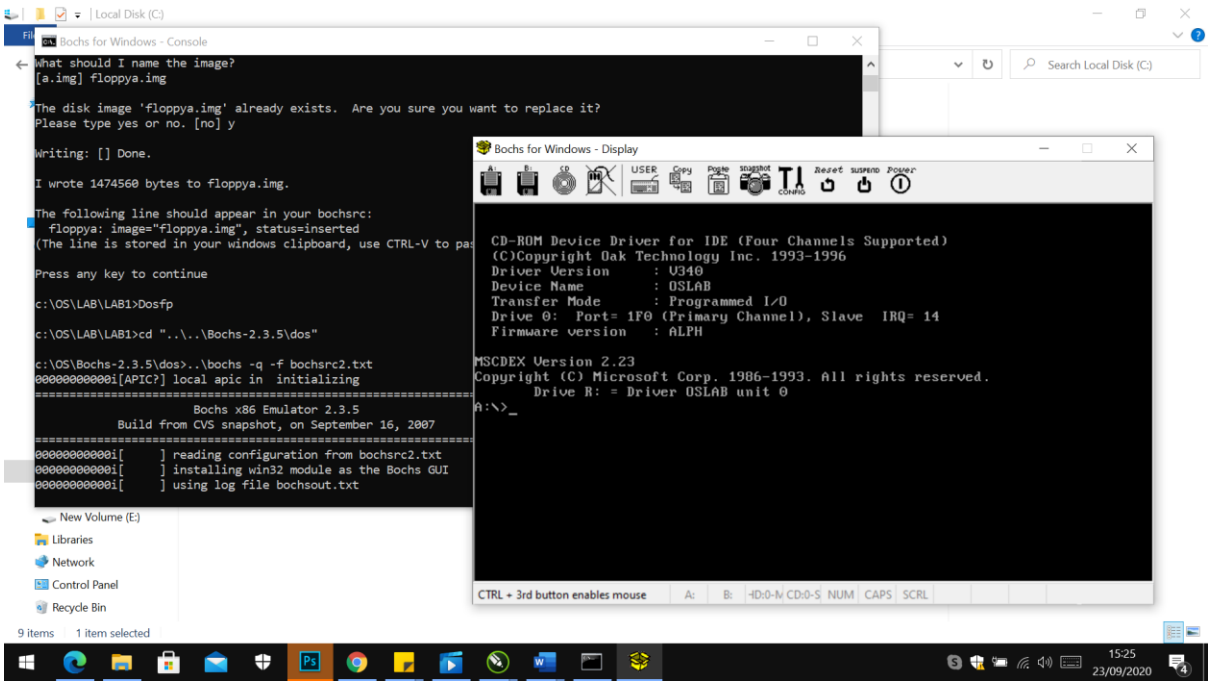


12. Terakhir memberikan nama image tersebut, berikan nama " floppya.img ", kemudian ketik " y ".

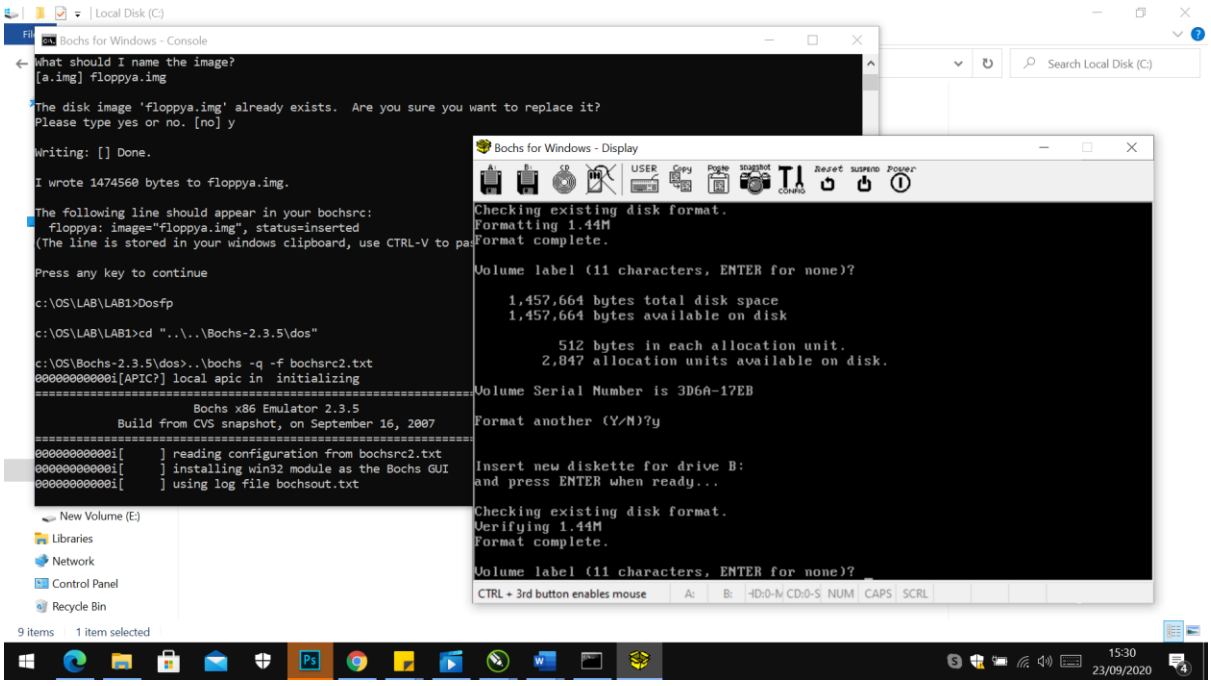


File image floppy 'floppya.img' yang dibuat dengan langkah di atas belum dapat digunakan, karena belum di 'FORMAT', seperti pada floppy sebenarnya jika belum di format floppy tidak dapat di baca. Langkah –langkah untuk memformat 'floppya.img' adalah:

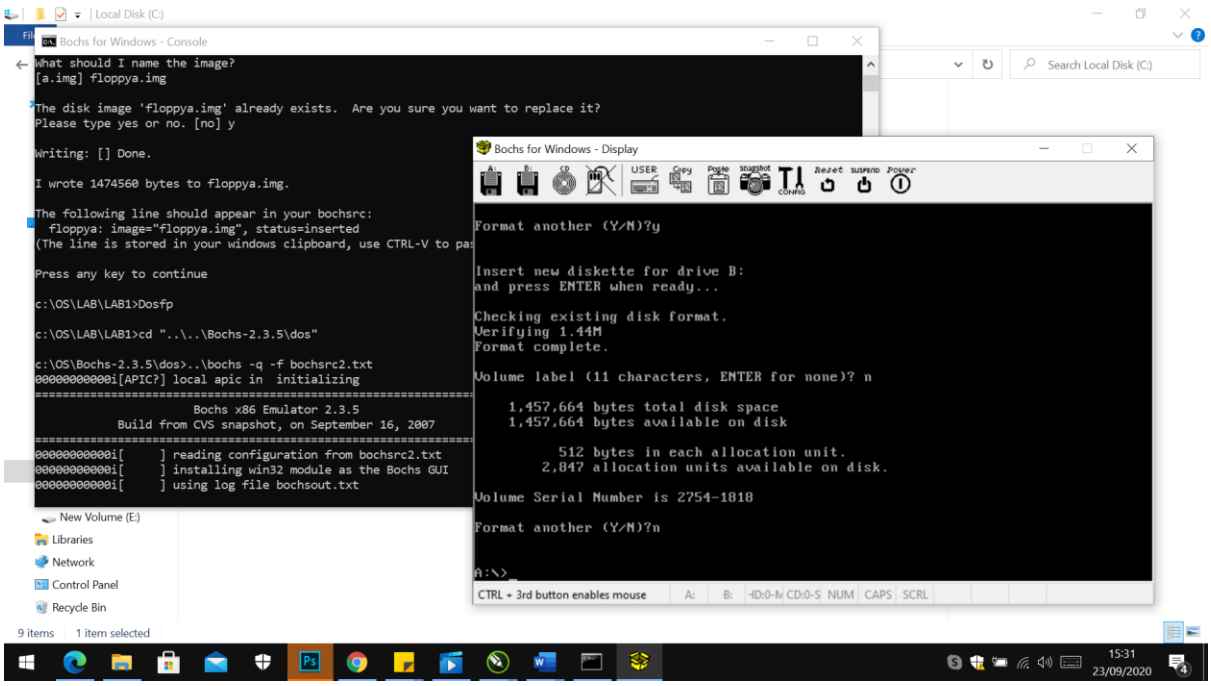
1. Ketikkan perintah “Dosfp” untuk memunculkan konfigurasi PC-Simulator



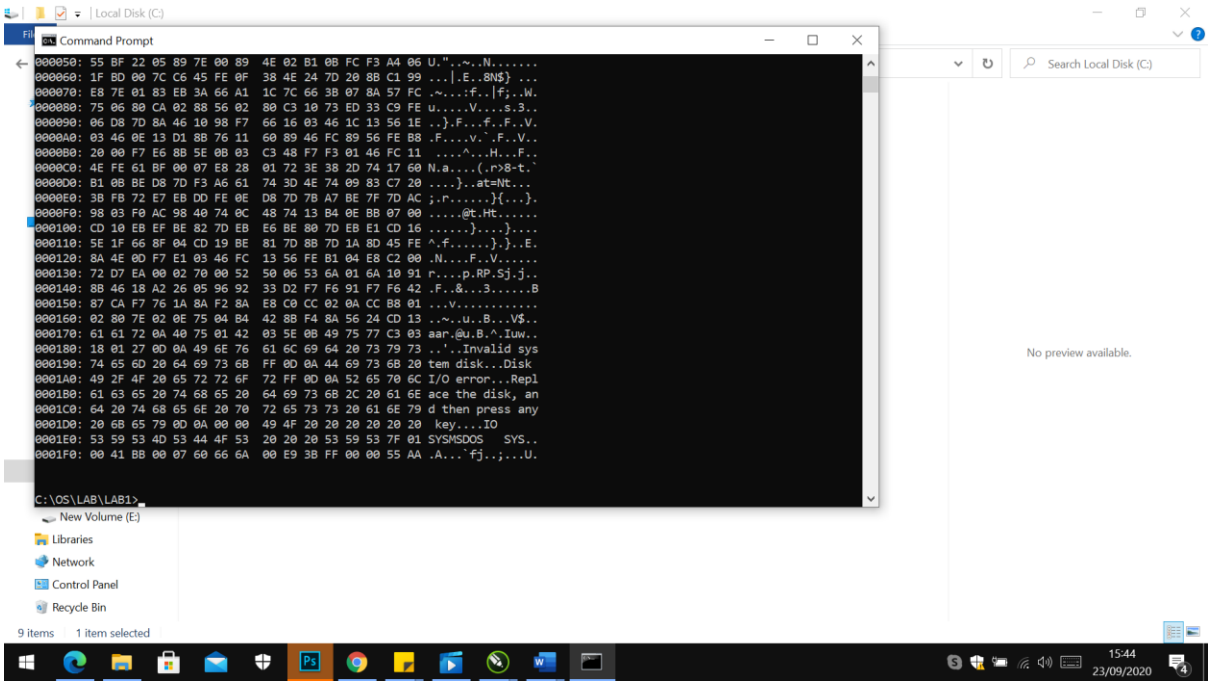
2. Kemudian dari prompt “ A:\ >Format B “, selanjutnya Enter , ketik “ y “, Enter,



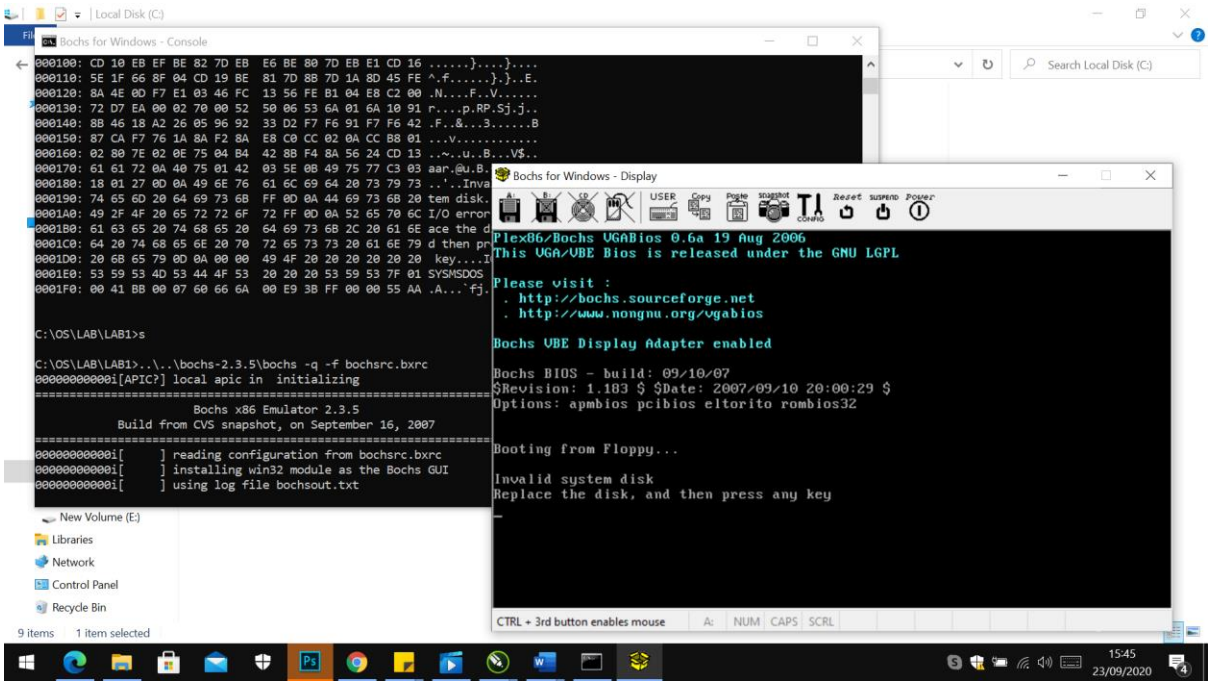
3. Sesudah menunjukkan complete kemudian ketik “ n ”. Lalu tutup PC Simulator dengan klik tombol power.



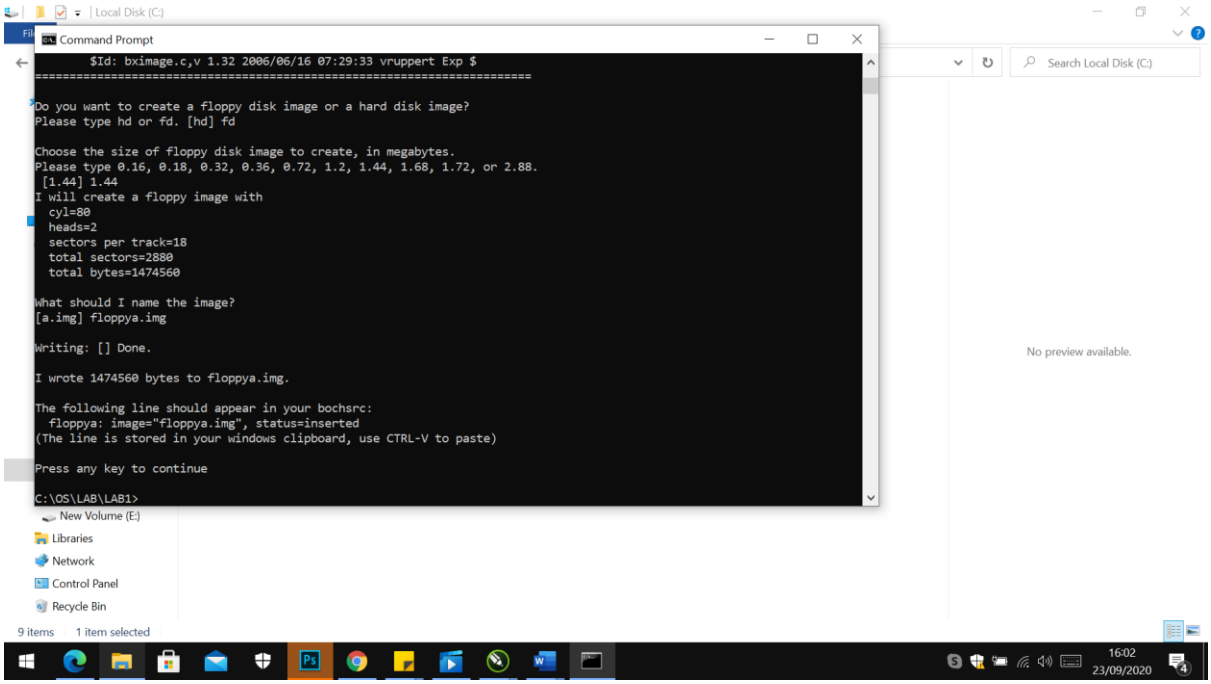
4. Sama dengan fungsi perintah “ debug ”, “ t dump boots.bin ” digunakan untuk memindah data dalam file ‘boots.bin’ ke dalam memori kerja ‘debug’ mulai dari alamat ‘0000:0100’. Perbedaan terletak pada kolom paling kiri, dengan ‘tdump’ kolom paling kiri menunjukkan posisi relatif setia byte data dimulai dari posisi ‘000000’.



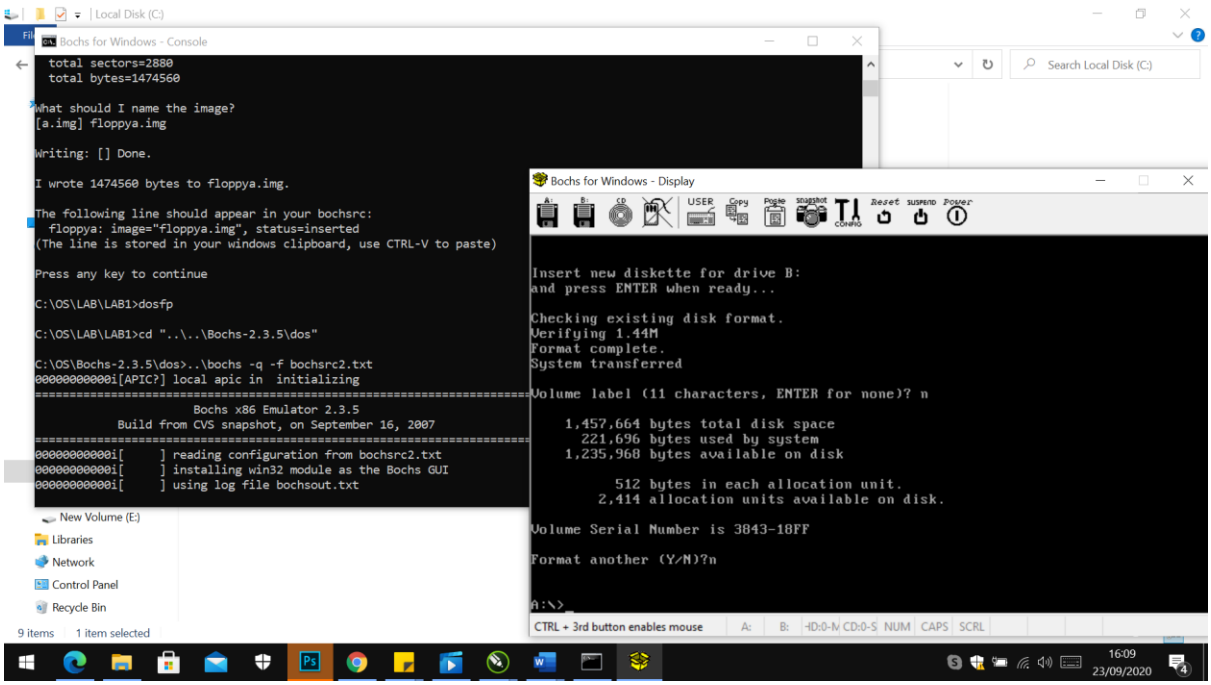
5. Untuk memanggil PC-Simulator, ketikkan perintah “ s ”. Namun tidak berhasil karena tidak menemukan diskboot.



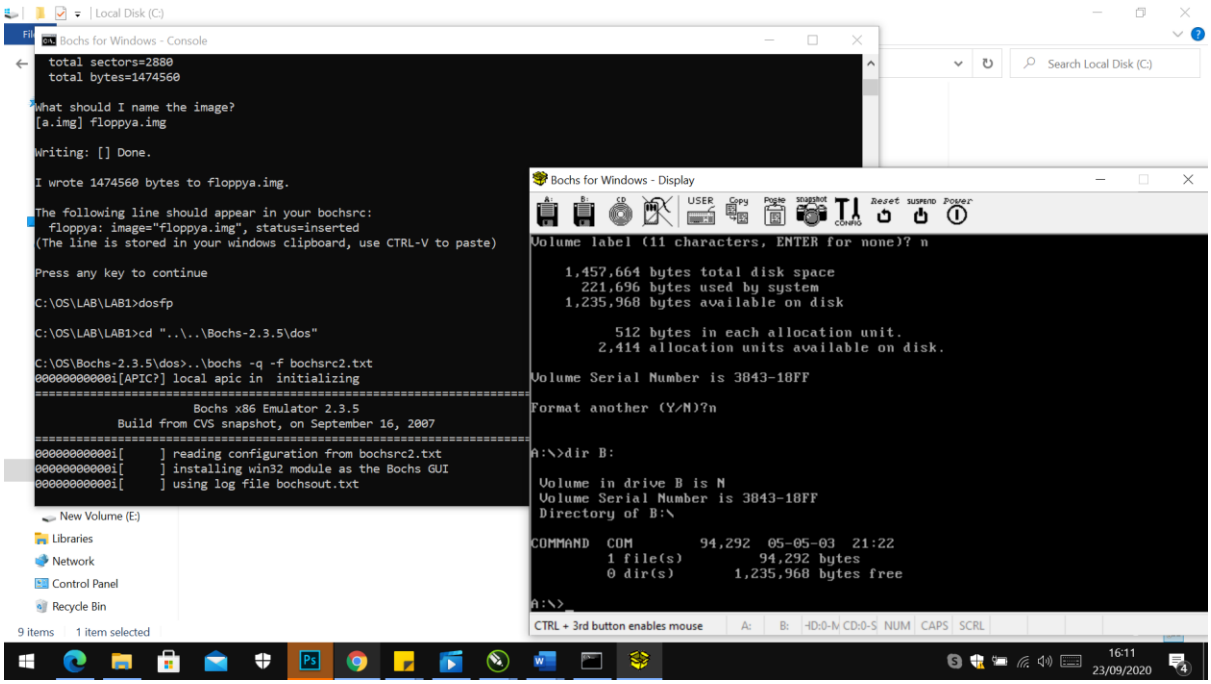
6. ‘Floppya.img’ belum terisi ‘system file’ atau ‘kernel’ sehingga proses boot gagal. Lakukan Langkah dibawah :
    - Hapus floppya.img (“ del floppya.img ”)
    - Buat file image floppy baru (“ bximage ”, pilih fd dengan type 1.44MB, beri nama floppya.img)



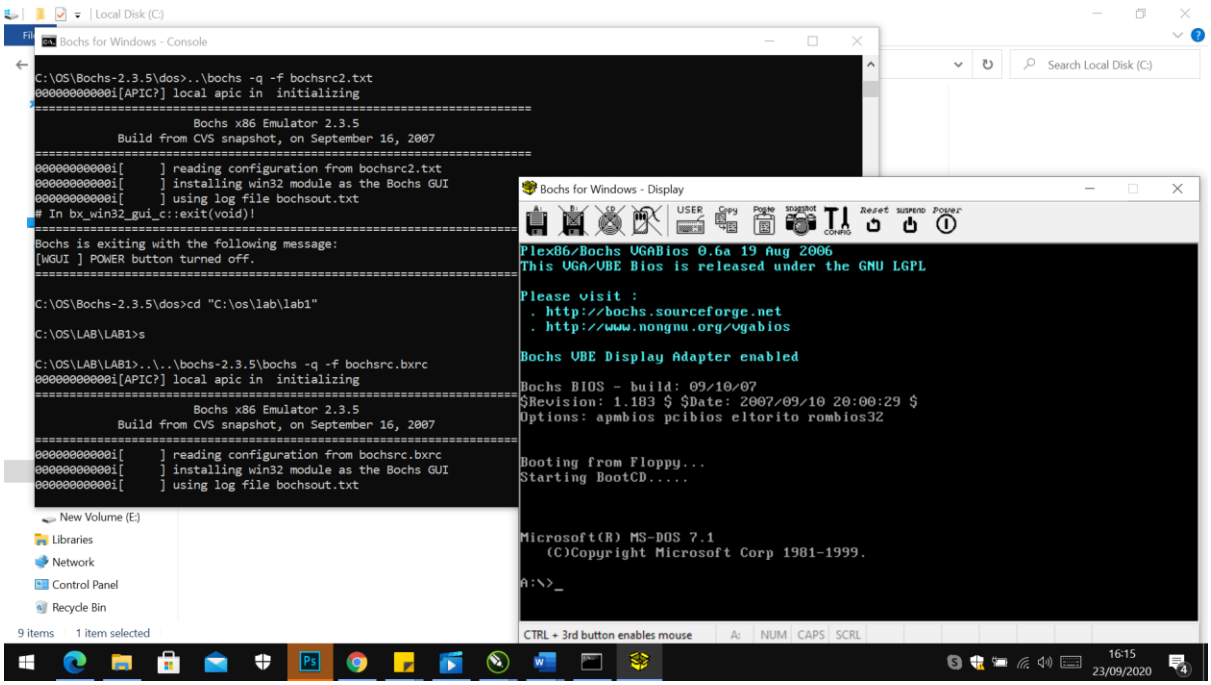
7. Setelah membuat file floppya.img baru. Sekarang tambahkan ‘system file’ ke dalamnya. Panggil ‘DosFp’, pada windows ‘Bochs’ masukan perintah ‘A:>format B:/S ’ , tekan Enter.



8. Untuk memastikan bahwa floppy pada drive ‘B:’ terisi dengan ‘system file’, periksa dengan perintah berikut “ A:>dir B: ”, lalu Enter.



9. Matikan PC-Simulator dengan klik power, coba untuk menggunakan ‘floppya.img’ sebagai “ boot disk “. Ketikkan “ s ”



Pada baris kedua terdapat teks “ starting BootCD “ Ini disebabkan oleh pembuatan boot disk awal menggunakan sistim yang berasal dari CD. Namun proses boot sebenarnya saat ini berasal dari ‘Floppya.img’.

Tugas Ebook Modul 1

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.

   Jawab :

   ASCII (American Standard Code for Information Interchange) merupakan Kode Standar Amerika untuk Pertukaran Informasi atau sebuah standar internasional dalam pengkodean huruf dan simbol seperti Unicode dan Hex tetapi ASCII lebih bersifat universal.

Tabel ASCII :

Binary	Oct	Dec	Hex	Glyph
010 0000	040	32	20	sr
010 0001	041	33	21	!
010 0010	042	34	22	"
010 0011	043	35	23	#
010 0100	044	36	24	$
010 0101	045	37	25	%
010 0110	046	38	26	&
010 0111	047	39	27	'
010 1000	050	40	28	(
010 1001	051	41	29	)
010 1010	052	42	2A	*
010 1011	053	43	2B	+
010 1100	054	44	2C	,
010 1101	055	45	2D	-
010 1110	056	46	2E	.
010 1111	057	47	2F	/
011 0000	060	48	30	0
011 0001	061	49	31	1
011 0010	062	50	32	2
011 0011	063	51	33	3
011 0100	064	52	34	4
011 0101	065	53	35	5
011 0110	066	54	36	6
011 0111	067	55	37	7
011 1000	070	56	38	8
011 1001	071	57	39	9
011 1010	072	58	3A	:
011 1011	073	59	3B	;
011 1100	074	60	3C	<
011 1101	075	61	3D	=
011 1110	076	62	3E	>
011 1111	077	63	3F	?

Binary	Oct	Dec	Hex	Glyph
100 0000	100	64	40	@
100 0001	101	65	41	A
100 0010	102	66	42	B
100 0011	103	67	43	C
100 0100	104	68	44	D
100 0101	105	69	45	E
100 0110	106	70	46	F
100 0111	107	71	47	G
100 1000	110	72	48	H
100 1001	111	73	49	I
100 1010	112	74	4A	J
100 1011	113	75	4B	K
100 1100	114	76	4C	L
100 1101	115	77	4D	M
100 1110	116	78	4E	N
100 1111	117	79	4F	O
101 0000	120	80	50	P
101 0001	121	81	51	Q
101 0010	122	82	52	R
101 0011	123	83	53	S
101 0100	124	84	54	T
101 0101	125	85	55	U
101 0110	126	86	56	V
101 0111	127	87	57	W
101 1000	130	88	58	X
101 1001	131	89	59	Y
101 1010	132	90	5A	Z
101 1011	133	91	5B	[
101 1100	134	92	5C	\
101 1101	135	93	5D	]
101 1110	136	94	5E	^
101 1111	137	95	5F	_

Binary	Oct	Dec	Hex	Glyph
110 0000	140	96	60	`
110 0001	141	97	61	a
110 0010	142	98	62	b
110 0011	143	99	63	c
110 0100	144	100	64	d
110 0101	145	101	65	e
110 0110	146	102	66	f
110 0111	147	103	67	g
110 1000	150	104	68	h
110 1001	151	105	69	i
110 1010	152	106	6A	j
110 1011	153	107	6B	k
110 1100	154	108	6C	l
110 1101	155	109	6D	m
110 1110	156	110	6E	n
110 1111	157	111	6F	o
111 0000	160	112	70	p
111 0001	161	113	71	q
111 0010	162	114	72	r
111 0011	163	115	73	s
111 0100	164	116	74	t
111 0101	165	117	75	u
111 0110	166	118	76	v
111 0111	167	119	77	w
111 1000	170	120	78	x
111 1001	171	121	79	y
111 1010	172	122	7A	z
111 1011	173	123	7B	{
111 1100	174	124	7C	|
111 1101	175	125	7D	}
111 1110	176	126	7E	~

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’.

No	Perintah	Penjelasan
1.	Komentar	Komentar diawali dengan tanda titik koma (;).
2.	Label	Label diakhiri dengan tanda titik dua (:)
3.	Assembler directives	
	a. Directives	perintah yang ditujukan kepada assembler ketika sedang menerjemahkan program kita ke bahasa mesin. Directive dimulai dengan tanda titik
	b. .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.
	c. .data	memberitahu assembler bahwa bagian di bawah ini adalah data program.
	d. .code	memberitahu assembler bahwa bagian di bawah ini adalah data program.
	e. .stack	memberitahu assembler bahwa program kita memiliki stack.
4.	Definisi data	
	a. DB : Define Bytes	Membentuk data byte demi byte. Data bisa data numerik maupun teks.
	b. DW : Define Words	Membentuk data word demi word (1 word = 2 byte).
	c. DD : Define Double Words	Membentuk data doubleword demi doubleword (4 byte).
	d. EQU : Equals	Membentuk konstanta.
5.	Perpindahan data	

	a. MOV : Move	Memindahkan suatu nilai dari register ke memori, memori ke register, atau register ke register.
	b. LEA : Load Effective Address	Mengisi suatu register dengan alamat offset sebuah data.
	c. XCHG : Exchange	Menukar dua buah register langsung.
	d. Lainnya :	IN, OUT, LODS, LODSB, LODSW, MOVS, MOVSB, MOVSW, LDS, LES, LAHF, SAHF, dan XLAT.
6.	Operasi logika	
	a. AND	melakukan bitwise AND.
	b. OR	melakukan bitwise OR.
	c. NOT	Melakukan bitwise NOT.
	d. XOR	Melakukan bitwise XOR.
	e. SHL : shift left	Menggeser bit ke kiri. Bit paling kanan diisi nol.
	f. SHR : shift right	Menggeser bit ke kanan. Bit paling kiri diisi nol.
	g. ROL : rotate left	Memutar bit ke kiri. Bit paling kiri jadi paling kanan kali ini.
	h. ROR:rotate right	Memutar bit ke kanan. Bit paling kanan jadi paling kiri.
	i. Lainnya	RCL dan RCR.
7.	Operasi matematika	
	a. ADD : Add	Menjumlahkan dua buah register.
	b. ADC : Add With Carry	Menjumlahkan dua register dan carry flag (CF).
	c. INC : Increment	Menjumlah isi sebuah register dengan 1.
	d. SUB : Substract	Mengurangkan dua buah register.

	e. SBB : Substract With Borrow	Mengurangkan dua register dan carry flag (CF).
	f. DEC:Decrement.	Mengurang isi sebuah register dengan 1.
	g. MUL : Multiply.	Mengalikan register dengan AX atau AH.
	h. IMUL : Signed Multiply.	Sama dengan MUL, hanya saja IMUL menganggap bit-bit yang ada di register sumber sudah dalam bentuk two's complement.
	i. DIV : Divide	Membagi AX atau DX:AX dengan sebuah register.
	j. IDIV: Signed Divide	Sama dengan DIV, hanya saja IDIV menganggap bit-bit yang ada di register sumber sudah dalam bentuk two's complement.
	k. NEG : Negate	Membuat isi register menjadi negatif (two's complement).
8.	Pengulangan	
	a. 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.
	b. LOOPE : Loop While Equal	Melakukan pengulangan selama CX ≠ 0 dan ZF = 1. CX tetap dikurangi 1 sebelum diperiksa.
	c. LOOPZ : Loop While Zero	Identik dengan LOOPE.
	d. LOOPNE : Loop While Not Equal	Melakukan pengulangan selama CX ≠ 0 dan ZF = 0. CX tetap dikurangi 1 sebelum diperiksa.

	e. LOOPNZ : Loop While Not Zero	Identik dengan LOOPNE.
	f. REP : repeat	Mengulang perintah sebanyak CX kali.
	g. REPE : Repeat While Equal	Mengulang perintah sebanyak CX kali, tetapi pengulangan segera dihentikan bila didapati ZF = 1
	h. REPZ : Repeat While Zero	Identik dengan REPE.
	i. REPNE : Repeat While Not Equal	Mengulang perintah sebanyak CX kali, tetapi pengulangan segera dihentikan bila didapati ZF = 0.
	j. REPNZ : Repeat While Not Zero	Identik dengan REPNE
9.	Perbandingan	
	a. CMP : Compare	Membandingkan dua buah operand. Hasilnya mempengaruhi sejumlah flag register.
10.	Lompat-lompat	
	a. JMP: Jump	Lompat tanpa syarat. Lompat begitu saja.
	b. Lompat bersyarat	Sintaksnya sama dengan JMP, yaitu perintah jump diikuti label tujuan.

		<table><tr><th>Perintah</th><th>Arti</th><th>Syarat</th><th>Kasus</th><th>Keterangan (“op = operand)</th></tr><tr><td>JA</td><td>jump if above</td><td rowspan="2">CF = 0 ∧ ZF = 0</td><td rowspan="2">unsigned</td><td rowspan="2">lompat bila op 1 > op 2</td></tr><tr><td>JNBE</td><td>jump if not below or equal</td></tr><tr><td>JB</td><td>jump if below</td><td rowspan="2">CF = 1 ∧ ZF = 0</td><td rowspan="2">unsigned</td><td rowspan="2">lompat bila op 1 < op 2</td></tr><tr><td>JNAE</td><td>jump if not above or equal</td></tr><tr><td>JAE</td><td>jump if above or equal</td><td rowspan="2">CF = 0 ∨ ZF = 1</td><td rowspan="2">unsigned</td><td rowspan="2">lompat bila op 1 ≥ op 2</td></tr><tr><td>JNB</td><td>jump if not below</td></tr><tr><td>JBE</td><td>jump if below or equal</td><td rowspan="2">CF = 1 ∨ ZF = 1</td><td rowspan="2">unsigned</td><td rowspan="2">lompat bila op 1 ≤ op 2</td></tr><tr><td>JNA</td><td>jump if not above</td></tr><tr><td>JG</td><td>jump if greater</td><td rowspan="2">OF = 0 ∧ ZF = 0</td><td rowspan="2">signed</td><td rowspan="2">lompat bila op 1 > op 2</td></tr><tr><td>JNLE</td><td>jump if not less or equal</td></tr><tr><td>JGE</td><td>jump if greater or equal</td><td rowspan="2">OF = 0 ∨ ZF = 1</td><td rowspan="2">signed</td><td rowspan="2">lompat bila op 1 ≥ op 2</td></tr><tr><td>JNL</td><td>jump if not less than</td></tr><tr><td>JL</td><td>jump if less than</td><td rowspan="2">OF = 1 ∧ ZF = 0</td><td rowspan="2">signed</td><td rowspan="2">lompat bila op 1 < op 2</td></tr><tr><td>JNGE</td><td>jump if not greater or equal</td></tr></table>
11.	Operasi stack	
	a. PUSH : Push	Menambahkan sesuatu ke stack.
	b. POP : Pop	Mengambil sesuatu dari stack.
	c. PUSHF : Push Flags	Mem-push semua isi register flag ke dalam stack. Biasa dipakai untuk membackup data di register flag sebelum operasi matematika.
	d. POPF : Pop Flags	Lawan dari pushf.
	e. 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.

	f. 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
12.	Operasi pada register flag	
	a. CLC : Clear Carry Flag	Menjadikan CF = 0.
	b. STC : Set Carry Flag	Menjadikan CF = 1.
	c. CMC: Complement Carry Flag	Melakukan operasi NOT pada CF. Yang tadinya 0 menjadi 1, dan sebaliknya.
	d. CLD : Clear Direction Flag	Menjadikan DF = 0.
	e. STD : Set Direction Flag	Menjadikan DF = 1.
	f. 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
	g. STI : Set Interrupt Flag	Menjadikan IF = 1.
13.	Perintah lainnya	
	a. ORG : Origin	Mengatur awal dari program (bagian static data). Analoginya seperti mengatur dimana letak titik (0, 0) pada koordinat Cartesius.

	b. 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.
	c. IRET : Interrupt-Handler Return.	Dapat 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.
	d. CALL : Call Procedure	Memanggil sebuah prosedur.
	e. RET : Return	Tanda selesai prosedur. Setiap prosedur harus memiliki RET di ujungnya.
	f. HLT : Halt	Membuat prosesor menjadi tidak aktif. Prosesor harus mendapat interupsi dari luar atau di-reset supaya aktif kembali. Jangan gunakan perintah HLT untuk mengakhiri program.
	g. NOP : No Operation	Perintah ini memakan 1 byte di memori tetapi tidak menyuruh prosesor melakukan apa-apa selama 3 clock prosesor.

Referensi :

https://oferiachacha.blogspot.com/2012/11/bahasa-assembly-untuk-x86.html

https://blog.ub.ac.id/fauziahmayasari/2012/07/18/bahasa-assembly-x86/