

**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**

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

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

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

sintaks: SUB {tujuan}. {sumber} operasi yang terjadi: tujuan = tujuan – sumber.  
borrow (bila terjadi) menyebabkan CF bernilai 1.

**SBB** : subtract 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  $\neq 0$  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  $\neq 0$  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 TI CMP?
<b>JA</b>	jump if above				
<b>JNBE</b>	jump if not below or equal	CF=0 ZF=0	unsigned	lompat bila op 1 > op 2	ya
<b>JB</b>	jump if below				
<b>JNAE</b>	jump if not above or equal	CF=1 ZF=0	unsigned	lompat bila op 1 < op 2	ya
<b>JAE</b>	jump if above or equal				
<b>JNB</b>	jump if not below	CF=0 ZF=1	unsigned	lompat bila op 1 $\geq$ op 2	ya
<b>JBE</b>	jump if below or equal				
<b>JNA</b>	jump if not above	CF=1 ZF=1	unsigned	lompat bila op 1 $\leq$ op 2	ya
<b>JG</b>	jump if greater				
<b>JNLE</b>	jump if not less or equal	OF=0 ZF=0	signed	lompat bila op 1 > op 2	ya
<b>JGE</b>	jump if greater or equal				
<b>JNL</b>	jump if not less than	OF=0 ZF=1	signed	lompat bila op 1 $\geq$ op 2	ya
<b>JL</b>	jump if less than				
<b>JNGE</b>	jump if not greater or equal	OF=1 ZF=0	signed	lompat bila op 1 < op 2	ya
<b>JLE</b>	jump if less or equal				
<b>JNG</b>	jump if not greater	OF=1 ZF=1	signed	lompat bila op 1 $\leq$ op 2	ya
<b>JE</b>	jump if equal	ZF=1	kedua nya	lompat bila op 1 = op 2	ya
<b>JZ</b>	jump if zero	ZF=1	kedua nya	lompat bila op 1 = op 2	ya
<b>JNE</b>	jump if not equal	ZF=0	kedua nya	lompat bila op 1 $\neq$ op 2	ya
<b>JNZ</b>	jump if not zero	ZF=0	kedua nya	lompat bila op 1 $\neq$ op 2	ya
<b>JC</b>	jump if carry	CF=1	N/A	lompat bila carry flag = 1	tidak
<b>JNC</b>	jump if not carry	CF=0	N/A	lompat bila carry flag = 0	tidak
<b>JP</b>	jump on parity			lompat bila parity flag = 1	
	jump on			lompat bila	tidak

JPE	parity even	PF=1	N/A	bilangan genap	selalu
<b>JNP</b>	jump on not parity			lompat bila parity flag = 0	
<b>JPO</b>	jump on parity odd	PF=0	N/A	lompat bila bilangan ganjil	tidak selalu
<b>JO</b>	jump if overflow	OF=1	N/A	lompat bila overflow flag = 1	tidak
<b>JNO</b>	jump if not overflow	OF=0	N/A	lompat bila overflow flag = 0	tidak
<b>JS</b>	jump if sign	SF=1	N/A	lompat bila bilangan negatif	tidak
<b>JCXZ</b>	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 kodennya: mov AX, seg teks1 push AX pop DS

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

Biasa dipakai untuk membackup 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 risikan 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 program 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*

