TUGAS PRAKTIKUM SISTEM OPERASI
MODUL 1
PENGENALAN SISTEM PENGEMBANGAN OS
DENGAN PC SIMULATOR 'BOCHS'

NAMA : NISA FATIMATUZ ZAHRO
NIM : L200190233
KELAS : F

PRODI INFORMATIKA
FAKULTAS KOMUNIKASI DAN INFORMATIKA
UNIVERSITAS MUHAMMADIYAH SURAKARTA

Tugas

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.

Binary	Oct	Dec	Hex	Glyph	Binary	Oct	Dec	Hex	Glyph	Binary	Oct	Dec	Hex	Glyph
010 0000	040	32	20	SP	100 0000	100	64	40	@	110 0000	140	96	60	`
010 0001	041	33	21	!	100 0001	101	65	41	A	110 0001	141	97	61	a
010 0010	042	34	22	"	100 0010	102	66	42	B	110 0010	142	98	62	b
010 0011	043	35	23	#	100 0011	103	67	43	C	110 0011	143	99	63	c
010 0100	044	36	24	$	100 0100	104	68	44	D	110 0100	144	100	64	d
010 0101	045	37	25	%	100 0101	105	69	45	E	110 0101	145	101	65	e
010 0110	046	38	26	&	100 0110	106	70	46	F	110 0110	146	102	66	f
010 0111	047	39	27	'	100 0111	107	71	47	G	110 0111	147	103	67	g
010 1000	050	40	28	(	100 1000	110	72	48	H	110 1000	150	104	68	h
010 1001	051	41	29	)	100 1001	111	73	49	I	110 1001	151	105	69	i

010	05	42	2A	*	100	11	74	4A	J	110	15	10	6A	j
1010	2				1010	2				1010	2	6		
010	05	43	2B	+	100	11	75	4B	K	110	15	10	6B	k
1011	3				1011	3				1011	3	7		
010	05	44	2C	,	100	11	76	4C	L	110	15	10	6C	l
1100	4				1100	4				1100	4	8		
010	05	45	2D	-	100	11	77	4D	M	110	15	10	6D	m
1101	5				1101	5				1101	5	9		
010	05	46	2E	.	100	11	78	4E	N	110	15	11	6E	n
1110	6				1110	6				1110	6	0		
010	05	47	2F	/	100	11	79	4F	O	110	15	11	6F	o
1111	7				1111	7				1111	7	1		
011	06	48	30	0	101	12	80	50	P	111	16	11	70	p
0000	0				0000	0				0000	0	2		
011	06	49	31	1	101	12	81	51	Q	111	16	11	71	q
0001	1				0001	1				0001	1	3		
011	06	50	32	2	101	12	82	52	R	111	16	11	72	r
0010	2				0010	2				0010	2	4		
011	06	51	33	3	101	12	83	53	S	111	16	11	73	s
0011	3				0011	3				0011	3	5		
011	06	52	34	4	101	12	84	54	T	111	16	11	74	t
0100	4				0100	4				0100	4	6		
011	06	53	35	5	101	12	85	55	U	111	16	11	75	u
0101	5				0101	5				0101	5	7		
011	06	54	36	6	101	12	86	56	V	111	16	11	76	v
0110	6				0110	6				0110	6	8		
011	06	55	37	7	101	12	87	57	W	111	16	11	77	w

0111	7				0111	7				0111	7	9		
011 1000	07 0	56	38	8	101 1000	13 0	88	58	X	111 1000	17 0	12 0	78	x
011 1001	07 1	57	39	9	101 1001	13 1	89	59	Y	111 1001	17 1	12 1	79	y
011 1010	07 2	58	3A	:	101 1010	13 2	90	5A	Z	111 1010	17 2	12 2	7A	z
011 1011	07 3	59	3B	;	101 1011	13 3	91	5B	[	111 1011	17 3	12 3	7B	{
011 1100	07 4	60	3C	<	101 1100	13 4	92	5C	\	111 1100	17 4	12 4	7C	|
011 1101	07 5	61	3D	=	101 1101	13 5	93	5D	]	111 1101	17 5	12 5	7D	}
011 1110	07 6	62	3E	>	101 1110	13 6	94	5E	^	111 1110	17 6	12 6	7E	~
011 1111	07 7	63	3F	?	101 1111	13 7	95	5F	_					

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 (;).

Contoh: ; 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 yaitu:

.186 .286 .286c .286p .287 .386 .386c .386p .387 .486 .486p .8086 .8087 .alpha .break .code .const .continue .cref .data .data? .dosseg .else .elseif .endif .endw .err .err1 .err2 .errb .errdef .errdif .errdifi .erre .erridn .erridni .errnb .errndef .errnz .exit .fardata .fardata? .if .lall .lfcond .list .listall .listif .listmacro .listmacroall .model .no87 .nocref .nolist .nolistif .nolistmacro .radix .repeat .sall .seq .sfcond .stack .startup .tfcond .type .until .untilcxz .while .xall .xcref .xlist.

<u>Definisi data</u>

- 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	CF = 0 ∧ ZF = 0	unsigned	lompat bila op 1 > op 2	ya
JNBE	jump if not below or equal				
JB	jump if below	CF = 1 ∧ ZF = 0	unsigned	lompat bila op 1 < op 2	ya
JNAE	jump if not above				

	or equal				
JAE	jump if above or equal	CF = 0 ∨ ZF = 1	unsigned	lompat bila op 1 ≥ op 2	ya
JNB	jump if not below				
JBE	jump if below or equal	CF = 1 ∨ ZF = 1	unsigned	lompat bila op 1 ≤ op 2	ya
JNA	jump if not above				
JG	jump if greater	OF = 0 ∧ ZF = 0	signed	lompat bila op 1 > op 2	ya
JNLE	jump if not less or equal				
JGE	jump if greater or equal	OF = 0 ∨ ZF = 1	signed	lompat bila op 1 ≥ op 2	ya
JNL	jump if not less than				
JL	jump if less than	OF = 1 ∧ ZF = 0	signed	lompat bila op 1 < op 2	ya
JNGE	jump if not greater or equal				
JLE	jump if less or equal	OF = 1 ∨ ZF = 1	signed	lompat bila op 1 ≤ op 2	ya
JNG	jump if				

	not greater				
JE	jump if equal	ZF = 1	keduanya	lompat bila op 1 = op 2	ya
JZ	jump if zero	ZF = 1	keduanya	lompat bila op 1 = op 2	ya
JNE	jump if not equal	ZF = 0	keduanya	lompat bila op 1 ≠ op 2	ya
JNZ	jump if not zero	ZF = 0	keduanya	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	PF = 1	N/A	lompat bila parity flag = 1	tidak selalu
JPE	jump on parity even			lompat bila bilangan genap	
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	CX = 0000	N/A	lompat bila CX berisi nol	tidak

	zero				

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).

- 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. 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.

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

Arsitektur x86

Ada ALU (aritmethic logic unit), ada register, ada I/O ke system bus, dan ada interkoneksi internal CPU itu sendiri.

Prosesor x86 memiliki 5 kelompok register, yaitu:

1. General Purpose Registers

Adalah register yang bisa dipakai untuk berbagai keperluan. Pada prosesor 8086 dan 286, register ini besarnya 16 bit. Register yang ada yaitu:

***AX** : accumulator register. Biasanya dipakai untuk menyimpan hasil hitungan matematika dan untuk menentukan service call.*

***BX** : base register. Biasanya dipakai untuk menunjuk indeks alamat memori.*

***CX** : counter register. Biasanya dipakai untuk pengulangan (loop).*

***DX** : data register. Biasanya dipakai untuk menyimpan data keluar/masuk prosesor, serta dipakai di operasi perkalian dan pembagian.*

Register generik ini masing-masing bisa “dipecah” menjadi dua, satu untuk MSB (high-order byte) dan satunya lagi untuk LSB (low-order byte).

Register AX bisa dipecah menjadi AH (AX high) dan AL (AX low), demikian juga ada BH, BL, CH, CL, DH, DL.

Sebagai contoh, bila AX sedang berisi 0110 0111 1101 0011(bin), maka AH berisi 0110 0111(bin) dan AL berisi 1101 0011(bin).

Untuk prosesor 386 ke atas (yang sudah 32 bit), terdapat register EAX (extended AX), EBX, ECX, dan EDX.

Masing-masing kapasitasnya 32 bit.

2. Segment Registers

Adalah register yang tugasnya mencatat blok memori (baca: segmen) yang sedang digunakan. Bila isinya diubah sembarangan, program bisa kacau. Register ini kaitannya dengan memori sementara :

CS *: code segment. Menunjuk segmen memori tempat kode program yang sekarang sedang jalan.*

DS *: data segment. Menunjuk segmen memori tempat data-data program disimpan (seperti pre-defined string,*

buffer string, dan konstanta-konstanta).

SS *: stack segment. Menunjuk segmen memori tempat "top" dari stack saat ini.*

"Segmen"nya, bukan "top"nya. Pembahasan tentang stack masih nanti.

ES *: extra segment. Adalah register "bonus" yang belum tentu dipakai, tergantung programnya. Misalnya untuk menunjuk alamat video memory pada program game yang fokus pada grafis.*

Pada prosesor 386 ke atas, ada tambahan extra segment lagi yang bernama **FS** dan **GS**. Huruf "F" dan "G" hanyalah urutan abjad sesudah "E" (ES), tidak ada arti khususnya.Semua register segmen, baik di prosesor 16-bit maupun 32-bit, kapasitasnya adalah 16-bit.

3. **Pointer Registers**

Adalah register yang berfungsi sebagai *pointer*.Isi dari register ini adalah alamat memori, makanya dia dikatakan "menunjuk" (*to point*) ke suatu alamat memori tertentu.

Kapasitas register pointer adalah 16 bit. Karena hanya 16 bit, maka tentu saja tidak semua alamat memori bisa ditunjuknya.

Besar memori yang mampu ditangani hanya 216 byte = 64 KB (saja), sangat kecil. Padahal prosesor 16 bit bisa menangani memori sampai 1 MB. Maka dari itu, register pointer ini bekerja berpasangan dengan register segmen menghasilkan alamat memori lengkap, dan mampu menangani sampai 1 MB.

***SP** : stack pointer. Menunjuk ke "top" dari stack-nya langsung.*

Operasi push dan pop akan mengubah isi dari SP. Penjelasan tentang stack masih nanti.

***BP** : base pointer. Menunjuk ke sebuah alamat di bagian data program.*

Bekerjasama dengan DS. Biasa dipakai untuk menunjuk elemen array.

***IP** : instruction pointer. Menunjuk ke alamat memori tempat instruksi berikutnya, di bagian kode program. Anggap saja IP adalah program counter (PC). Register ini diubah otomatis sejalan dengan jalannya program. Pada prosesor 386 ke atas, terdapat register ESP, EBP, dan EIP yang kapasitasnya 32 bit sehingga mampu menangani memori sampai 232 byte = 4 GB.*

4. **Index Registers**

Register ini digunakan oleh operasi string dan block transfer di memori.

***SI** : source index.*

***DI** : destination index.*

Kapasitasnya 16 bit. Pada prosesor 386 ke atas, terdapat ESI dan EDI yang kapasitasnya 32 bit.

5. **Flag Registers**

Namanya juga "bendera", register ini berfungsi menandakan suatu keadaan "ya" (mengibarkan bendera) atau "tidak" (tidak mengibarkan bendera). Tentu saja tidak ada bendera di dalam CPU Anda, yang ada hanyalah bit 1 atau 0. Register bendera ini masing-masing besarnya 1 bit saja.

***OF** : overflow register. Bila terjadi overflow pada operasi matematika, maka OF bernilai 1. Bila tidak, isi 0.*

***SF** : sign flag. Jika suatu operasi menghasilkan angka negatif, SF berisi 1.*

***ZF** : zero flag. Jika suatu operasi menghasilkan angka nol, ZF berisi 1.*

***CF** : carry flag. Berisi bit carry pada operasi penjumlahan atau bit borrow pada operasi pengurangan.*

***DF** : direction flag. Menunjukkan arah pembacaan byte (maju atau mundur) pada operasi string.*

***PF** : parity flag. Bila angka yang dihitung genap atau ganjil (tergantung sistemnya mau genap apa ganjil), PF berisi 1.*
***AF** : auxiliary flag. Berisi 1 setelah penjumlahan dua bilangan BCD. Fungsinya seperti carry flag (CF) setelah bit ke-4 (bukan ke-8)*
***TF** : trap flag. Menunjukkan mode debugging on atau off. Ini urusan internal CPU.*
***IF** : interrupt flag. Bila IF bernilai 0, maka interupsi ke prosesor akan diabaikan.*

Register yang hanya terdapat di 80286 ke atas:
NT : nested task.
IOPL : I/O protection level. (2 bit)
PE : protection enable.
MP : monitor co-processor.
EM : emulate co-processor.
TS : task switched.
ET : extention type.
RF : resume flag.
VF : virtual 8086 mode.