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

DANDUN GIGIH PRAKOSO

L200190172

FAKULTAS KOMUNIKASI DAN INFORMATIKA
UNIVERSITAS MUHAMMADIYAH SURAKARTA
2020

Tugas

1. Apa yang dimaksud dengan kode ‘ASCII’, buatlah table kode ASCII lengkap cukup kode ASCII yang standar tidak perlu extended, tuliskan kode ASCII dalam format angka decimal, binary dan hexa decimal serta karakter dan simbol yang dikodekan.
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’

Jawaban :

1. ASCII (American Standard Code for Information Interchange) atau kode standar amerika yang digunakan untuk pertukaran informasi , merupakan sebuah standar internasional dalam pengkodean huruf dan simbol (karakter) untuk alat komunikasi seperti Unicode dan Hex tetapi ASCII ini lebih bersifat universal.

Karakter	Hexadecimal	Decimal	Binary	Keterangan
NUL	0000	0	0	Null (tidak tampak)
SOH	0001	1	1	Start of heading (tidak tampak)
STX	0002	2	10	Start of text (tidak tampak)
ETX	0003	3	11	End of text (tidak tampak)
EOT	0004	4	100	End of transmission (tidak tampak)
ENQ	0005	5	101	Enquiry (tidak tampak)
ACK	0006	6	110	Acknowledge (tidak tampak)
BEL	0007	7	111	Bell (tidak tampak)
BS	0008	8	1000	Menghapus satu karakter di belakang kursor (Backspace)
HT	0009	9	1001	Horizontal tabulation
LF	000A	10	1010	Pergantian baris (Line feed)
VT	000B	11	1011	Tabulasi vertikal
FF	000C	12	1100	Pergantian baris (Form feed)
CR	000D	13	1101	Pergantian baris (carriage return)
SO	000E	14	1110	Shift out (tidak tampak)
SI	000F	15	1111	Shift in (tidak tampak)
DLE	0010	16	10000	Data link escape (tidak tampak)
DC1	0011	17	10001	Device control 1 (tidak tampak)
DC2	0012	18	10010	Device control 2 (tidak tampak)
DC3	0013	19	10011	Device control 3 (tidak tampak)
DC4	0014	20	10100	Device control 4 (tidak tampak)
NAK	0015	21	10101	Negative acknowledge (tidak tampak)
SYN	0016	22	10110	Synchronous idle (tidak tampak)
ETB	0017	23	10111	End of transmission block (tidak tampak)
CAN	0018	24	11000	Cancel (tidak tampak)
EM	0019	25	11001	End of medium (tidak tampak)
SUB	001A	26	11010	Substitute (tidak tampak)
ESC	001B	27	11011	Escape (tidak tampak)
FS	001C	28	11100	File separator
GS	001D	29	11101	Group separator
RS	001E	30	11110	Record separator
US	001F	31	11111	Unit separator
SP	0020	32	100000	Spasi
!	0021	33	100001	Tanda seru (exclamation)
"	0022	34	100010	Tanda kutip dua
#	0023	35	100011	Tanda pagar (kres)
$	0024	36	100100	Tanda mata uang dolar
%	0025	37	100101	Tanda persen
&	0026	38	100110	Karakter ampersand (&)
‘	0027	39	100111	Karakter Apostrof

(	0028	40	101000	Tanda kurung buka
)	0029	41	101001	Tanda kurung tutup
*	002A	42	101010	Karakter asterisk (bintang)
+	002B	43	101011	Tanda tambah (plus)
,	002C	44	101100	Karakter koma
-	002D	45	101101	Karakter hyphen (strip)
.	002E	46	101110	Tanda titik
/	002F	47	101111	Garis miring (*slash*)
0	0030	48	110000	Angka nol
1	0031	49	110001	Angka satu
2	0032	50	110010	Angka dua
3	0033	51	110011	Angka tiga
4	0034	52	110100	Angka empat
5	0035	53	110101	Angka lima
6	0036	54	110110	Angka enam
7	0037	55	110111	Angka tujuh
8	0038	56	111000	Angka delapan
9	0039	57	111001	Angka sembilan
:	003A	58	111010	Tanda titik dua
;	003B	59	111011	Tanda titik koma
<	003C	60	111100	Tanda lebih kecil
=	003D	61	111101	Tanda sama dengan
>	003E	62	111110	Tanda lebih besar
?	003F	63	111111	Tanda tanya
@	0040	64	1000000	A keong (@)
A	0041	65	1000001	Huruf latin A kapital
B	0042	66	1000010	Huruf latin B kapital
C	0043	67	1000011	Huruf latin C kapital
D	0044	68	1000100	Huruf latin D kapital
E	0045	69	1000101	Huruf latin E kapital
F	0046	70	1000110	Huruf latin F kapital
G	0047	71	1000111	Huruf latin G kapital
H	0048	72	1001000	Huruf latin H kapital
I	0049	73	1001001	Huruf latin I kapital
J	004A	74	1001010	Huruf latin J kapital
K	004B	75	1001011	Huruf latin K kapital
L	004C	76	1001100	Huruf latin L kapital
M	004D	77	1001101	Huruf latin M kapital
N	004E	78	1001110	Huruf latin N kapital
O	004F	79	1001111	Huruf latin O kapital
P	0050	80	1010000	Huruf latin P kapital
Q	0051	81	1010001	Huruf latin Q kapital
R	0052	82	1010010	Huruf latin R kapital
S	0053	83	1010011	Huruf latin S kapital
T	0054	84	1010100	Huruf latin T kapital
U	0055	85	1010101	Huruf latin U kapital
V	0056	86	1010110	Huruf latin V kapital
W	0057	87	1010111	Huruf latin W kapital
X	0058	88	1011000	Huruf latin X kapital
Y	0059	89	1011001	Huruf latin Y kapital
Z	005A	90	1011010	Huruf latin Z kapital
[	005B	91	1011011	Kurung siku kiri
\	005C	92	1011100	Garis miring terbalik (*backslash*)
]	005D	93	1011101	Kurung sikur kanan
^	005E	94	1011110	Tanda pangkat
_	005F	95	1011111	Garis bawah (underscore)

`	0060	96	1100000	Tanda petik satu
A	0061	97	1100001	Huruf latin a kecil
B	0062	98	1100010	Huruf latin b kecil
C	0063	99	1100011	Huruf latin c kecil
D	0064	100	1100100	Huruf latin d kecil
E	0065	101	1100101	Huruf latin e kecil
F	0066	102	1100110	Huruf latin f kecil
G	0067	103	1100111	Huruf latin g kecil
H	0068	104	1101000	Huruf latin h kecil
I	0069	105	1101001	Huruf latin i kecil
J	006A	106	1101010	Huruf latin j kecil
K	006B	107	1101011	Huruf latin k kecil
L	006C	108	1101100	Huruf latin l kecil
M	006D	109	1101101	Huruf latin m kecil
N	006E	110	1101110	Huruf latin n kecil
O	006F	111	1101111	Huruf latin o kecil
P	0070	112	1110000	Huruf latin p kecil
Q	0071	113	1110001	Huruf latin q kecil
R	0072	114	1110010	Huruf latin r kecil
S	0073	115	1110011	Huruf latin s kecil
T	0074	116	1110100	Huruf latin t kecil
U	0075	117	1110101	Huruf latin u kecil
V	0076	118	1110110	Huruf latin v kecil
W	0077	119	1110111	Huruf latin w kecil
X	0078	120	1111000	Huruf latin x kecil
Y	0079	121	1111001	Huruf latin y kecil
Z	007A	122	1111010	Huruf latin z kecil
{	007B	123	1111011	Kurung kurawal buka
¦	007C	124	1111100	Garis vertikal (pipa)
}	007D	125	1111101	Kurung kurawal tutup
~	007E	126	1111110	Karakter gelombang (tilde)
DEL	007F	127	1111111	Delete
	0080	128	10000000	Dicadangkan
	0081	129	10000001	Dicadangkan
	0082	130	10000010	Dicadangkan
	0083	131	10000011	Dicadangkan
IND	0084	132	10000100	Index
NEL	0085	133	10000101	Next line
SSA	0086	134	10000110	Start of selected area
ESA	0087	135	10000111	End of selected area
	0088	136	10001000	Character tabulation set
	0089	137	10001001	Character tabulation with justification
	008A	138	10001010	Line tabulation set
PLD	008B	139	10001011	Partial line down
PLU	008C	140	10001100	Partial line up
	008D	141	10001101	Reverse line feed
SS2	008E	142	10001110	Single shift two
SS3	008F	143	10001111	Single shift three
DCS	0090	144	10010000	Device control string
PU1	0091	145	10010001	Private use one
PU2	0092	146	10010010	Private use two
STS	0093	147	10010011	Set transmit state
CCH	0094	148	10010100	Cancel character
MW	0095	149	10010101	Message waiting
	0096	150	10010110	Start of guarded area
	0097	151	10010111	End of guarded area

2. Daftar perintah Bahasa assembly untuk mesin intel keluarga x86 :

Definisi data

1. **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)
2. **DD** : define double words. Membentuk data doubleword demi doubleword (4 byte).
   sintaks: {label} DD {data} **EQU** : equals. Membentuk konstanta. sintaks: {label} E {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

1. **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.

2. **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

1. **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.
2. **OR** : melakukan bitwise or. sintaks: OR {register}, {angka} OR {register 1}, {register 2} hasil disimpan di register 1.
3. **NOT** : melakukan bitwise not (*one's complement*) sintaks: NOT {register} hasil disimpan di register itu sendiri.
4. **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.
5. **SHL** : shift left. Menggeser bit ke kiri. Bit paling kanan diisi nol. sintaks: SHL {register}, {banyaknya}
6. **SHR** : shift right. Menggeser bit ke kanan. Bit paling kiri diisi nol. sintaks: SHR {register}, {banyaknya}
7. **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.
8. **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

1. **ADD** : add. Menjumlahkan dua buah register.
   sintaks: ADD {tujuan}, {sumber} operasi yang terjadi: tujuan = tujuan + sumber.
   carry (bila ada) disimpan di CF.
2. **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.
3. **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}
4. **SUB** : substract. Mengurangkan dua buah register.
   sintaks: SUB {tujuan}. {sumber} operasi yang terjadi: tujuan = tujuan – sumber.

borrow (bila terjadi) menyebabkan CF bernilai 1.

5. **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.
6. **DEC** : decrement. Mengurang isi sebuah register dengan 1.
   Jika SUB memakai 3 byte memori, DEC hanya memakai 1 byte. sintaks: DEC {register}
7. **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.
8. **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}
9. **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.
10. **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}
11. **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

1. **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).
2. **LOOPE** : loop while equal. Melakukan pengulangan selama CX ≠ 0 dan ZF = 1. CX tetap dikurangi 1 sebelum diperiksa.
   sintaks: LOOP {label tujuan}
3. **LOOPZ** : loop while zero. Identik dengan LOOPE.
4. **LOOPNE** : loop while not equal.

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

5. **LOOPNZ** : loop while not zero. Identik dengan LOOPNE.
6. **REP** : repeat. Mengulang perintah sebanyak CX kali. sintaks: REP {perintah assembly}
   contoh:
   mov CX, 05 rep inc BX ;register BX ditambah 1 sebanyak 5x.
7. **REPE** : repeat while equal. Mengulang perintah sebanyak CX kali, tetapi pengulangan segera dihentikan bila didapati ZF = 1.
   sintaks: REPE {perintah assembly}
8. **REPZ** : repeat while zero. Identik dengan REPE.
9. **REPNE** : repeat while not equal. Mengulang perintah sebanyak CX kali, tetapi pengulangan segera dihentikan bila didapati ZF = 0.
   sintaks: REPNE {perintah assembly}
10. **REPNZ** : repeat while not zero. Identik dengan REPNE.

Perbandingan

1. **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

1. **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 zero	CX = 0000	N/A	lompat bila CX berisi nol	tidak

Operasi stack

1. **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".
2. **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
3. **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).
4. **POPF** : pop flags. Lawan dari pushf. Sintaks: POPF ;(saja).
5. **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.
6. **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

1. **CLC** : clear carry flag. Menjadikan CF = 0. Sintaks: CLC ;(saja).
2. **STC** : set carry flag. Menjadikan CF = 1. Sintaks: STC ;(saja).
3. **CMC** : complement carry flag. Melakukan operasi NOT pada CF. Yang tadinya 0 menjadi 1, dan sebaliknya.
4. **CLD** : clear direction flag. Menjadikan DF = 0. Sintaks: CLD ;(saja).
5. **STD** : set direction flag. Menjadikan DF = 1.
6. **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.
7. **STI** : set interrupt flag. Menjadikan IF = 1.
   Perintah lainnya
8. **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).
9. **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}
10. **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.
11. **CALL** : call procedure. Memanggil sebuah prosedur.
    sintaks: CALL {label nama prosedur}
12. **RET** : return. Tanda selesai prosedur.
    Setiap prosedur harus memiliki RET di ujungnya.
    sintaks: RET ;(saja)
13. **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