Nama :Herlanggga Yusuf Syailendra

NIM : L200180186

Module 1

1. Kode Standar Amerika untuk Pertukaran Informasi atau ASCII (American Standard Code for Information Interchange) adalah suatu standar internasional dalam kode huruf dan simbol seperti Hex dan Unicode, tetapi ASCII lebih bersifat universal. Karakter control pada ASCII dibedakan menjadi 5 kelompok sesuai dengan penggunaan yakni meliputi logical communication, device control, information separator, code extention, dan physical communication.

tabel ASCII Standard

Decimal	Octal	Hex	Binary	Value	Description
0	0	0	0000 0000	NUL	null character
1	1	1	0000 0001	SOH	start of header
2	2	2	0000 0010	STX	start of text
3	3	3	0000 0011	ETX	end of text
4	4	4	0000 0100	EOT	end of transmission
5	5	5	0000 0101	ENQ	enquiry
6	6	6	0000 0110	ACK	acknowledgment
7	7	7	0000 0111	BEL	bell
8	10	8	0000 1000	BS	backspace
9	11	9	0000 1001	HT	horizontal tab
10	12	0A	0000 1010	LF	line feed
11	13	0B	0000 1011	VT	vertical tab
12	14	0C	0000 1100	FF	form feed
13	15	0D	0000 1101	CR	carriage return
14	16	0E	0000 1110	SO	shift out
15	17	0F	0000 1111	SI	shift in
16	20	10	0001 0000	DLE	data link escape
17	21	11	0001 0001	DC1	device control 1 (XON)
18	22	12	0001 0010	DC2	device control 2
19	23	13	0001 0011	DC3	device control 3 (XOFF)

20	24	14	0001 0100	DC4	device control 4
21	25	15	0001 0101	NAK	negative acknowledgement
22	26	16	0001 0110	SYN	synchronous idle
23	27	17	0001 0111	ETB	end of transmission block
24	30	18	0001 1000	CAN	cancel
25	31	19	0001 1001	EM	end of medium
26	32	1A	0001 1010	SUB	substitute
27	33	1B	0001 1011	ESC	escape
28	34	1C	0001 1100	FS	file separator
29	35	1D	0001 1101	GS	group separator
30	36	1E	0001 1110	RS	request to send/record separator
31	37	1F	0001 1111	US	unit separator
32	40	20	0010 0000	SP	space
33	41	21	0010 0001	!	exclamation mark
35	43	23	0010 0011	#	number sign
36	44	24	0010 0100	$	dollar sign
37	45	25	0010 0101	%	percent
38	46	26	0010 0110	&	ampersand
39	47	27	0010 0111		single quote
40	50	28	0010 1000	(	left/opening parenthesis
41	51	29	0010 1001	)	right/closing parenthesis
42	52	2A	0010 1010	*	asterisk
43	53	2B	0010 1011	+	plus
44	54	2C	0010 1100	,	comma
45	55	2D	0010 1101	-	minus or dash
46	56	2E	0010 1110	.	dot
47	57	2F	0010 1111	/	forward slash
48	60	30	0011 0000	0	
49	61	31	0011 0001	1	
50	62	32	0011 0010	2	

51	63	33	0011 0011	3	
52	64	34	0011 0100	4	
53	65	35	0011 0101	5	
54	66	36	0011 0110	6	
55	67	37	0011 0111	7	
56	70	38	0011 1000	8	
57	71	39	0011 1001	9	
58	72	3A	0011 1010	:	colon
59	73	3B	0011 1011	;	semi-colon
60	74	3C	0011 1100	<	less than
61	75	3D	0011 1101	=	equal sign
62	76	3E	0011 1110	>	greater than
63	77	3F	0011 1111	?	question mark
64	100	40	0100 0000	@	at symbol
65	101	41	0100 0001	A	
66	102	42	0100 0010	B	
67	103	43	0100 0011	C	
68	104	44	0100 0100	D	
69	105	45	0100 0101	E	
70	106	46	0100 0110	F	
71	107	47	0100 0111	G	
72	110	48	0100 1000	H	
73	111	49	0100 1001	I	
74	112	4A	0100 1010	J	
75	113	4B	0100 1011	K	
76	114	4C	0100 1100	L	
77	115	4D	0100 1101	M	
78	116	4E	0100 1110	N	
79	117	4F	0100 1111	O	
80	120	50	0101 0000	P	

81	121	51	0101 0001	Q	
82	122	52	0101 0010	R	
83	123	53	0101 0011	S	
84	124	54	0101 0100	T	
85	125	55	0101 0101	U	
86	126	56	0101 0110	V	
87	127	57	0101 0111	W	
88	130	58	0101 1000	X	
89	131	59	0101 1001	Y	
90	132	5A	0101 1010	Z	
91	133	5B	0101 1011	[	left/opening bracket
92	134	5C	0101 1100	\	back slash
93	135	5D	0101 1101	]	right/closing bracket
94	136	5E	0101 1110	^	caret/circumflex
95	137	5F	0101 1111	_	underscore
96	140	60	0110 0000	`	
97	141	61	0110 0001	a	
98	142	62	0110 0010	b	
99	143	63	0110 0011	c	
100	144	64	0110 0100	d	
101	145	65	0110 0101	e	
102	146	66	0110 0110	f	
103	147	67	0110 0111	g	
104	150	68	0110 1000	h	
105	151	69	0110 1001	i	
106	152	6A	0110 1010	j	
107	153	6B	0110 1011	k	
108	154	6C	0110 1100	l	
109	155	6D	0110 1101	m	
110	156	6E	0110 1110	n	

111	157	6F	0110 1111	o	
112	160	70	0111 0000	p	
113	161	71	0111 0001	q	
114	162	72	0111 0010	r	
115	163	73	0111 0011	s	
116	164	74	0111 0100	t	
117	165	75	0111 0101	u	
118	166	76	0111 0110	v	
119	167	77	0111 0111	w	
120	170	78	0111 1000	x	
121	171	79	0111 1001	y	
122	172	7A	0111 1010	z	
123	173	7B	0111 1011	{	left/opening brace
124	174	7C	0111 1100 |		vertical bar
125	175	7D	0111 1101	}	right/closing brace
126	176	7E	0111 1110	~	tilde
127	177	7F	0111 1111	DEL	delete

2. Carilah Tabel Instruksi perintah bahasa Assembly X86

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 .elseif .endif .endw .err .err 1 .err2 .errb
.errdef .errdif .errdifi .erre .erridn .erridni .errnb .errndef .errnz .exit .fardata .fardata? .if .lall .lfc ond .list .listall .listif .listmacro
.listmacroall .model .no87 .nocref .nolist .nolistif .nolistmacro .radix .repeat .sall .seq .sfcond .st ack
.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	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

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