Nama : Alif Al Amin

NIM : L200180082

Kelas : B

MODUL 1

1**. Kode Standar Amerika untuk Pertukaran Informasi** atau *American Standard Code for Information Interchange* (**ASCII**) 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 7 bit. Namun, ASCII disimpan sebagai sandi 8 bit dengan menambakan satu angka 0 sebagai bit significant paling tinggi. Bit tambahan ini sering digunakan untuk uji paritas. Karakter control pada ASCII dibedakan menjadi 5 kelompok sesuai dengan penggunaan yaitu berturut-turut meliputi logical communication, Device control, Information separator, Code extention, dan physical communication. Code ASCII ini banyak dijumpai pada papan ketik (keyboard) computer atau instrument-instrument digital.

Berikut adalah table ASCII :

Karakter	Nilai Unicode (heksadesimal)	Nilai ANSI ASCII (desimal)	Binary	Keterangan
NUL	0000	0	00000000	Null (tidak tampak)
SOH	0001	1	00000001	Start of heading (tidak tampak)
STX	0002	2	00000010	Start of text (tidak tampak)
ETX	0003	3	00000011	End of text (tidak tampak)
EOT	0004	4	00000100	End of transmission (tidak tampak)
ENQ	0005	5	00000101	Enquiry (tidak tampak)
ACK	0006	6	00000110	Acknowledge (tidak tampak)
BEL	0007	7	00000111	Bell (tidak tampak)
BS	0008	8	00001000	Menghapus satu karakter di belakang kursor (Backspace)
HT	0009	9	00001001	Horizontal tabulation
LF	000A	10	00001010	Pergantian baris (Line feed)
VT	000B	11	00001011	Tabulasi vertikal
FF	000C	12	00001100	Pergantian baris (Form feed)
CR	000D	13	00001101	Pergantian baris (carriage return)

SO	000E	14	00001110	Shift out (tidak tampak)
SI	000F	15	00001111	Shift in (tidak tampak)
DLE	0010	16	00010000	Data link escape (tidak tampak)
DC1	0011	17	00010001	Device control 1 (tidak tampak)
DC2	0012	18	00010010	Device control 2 (tidak tampak)
DC3	0013	19	00010011	Device control 3 (tidak tampak)
DC4	0014	20	00010100	Device control 4 (tidak tampak)
NAK	0015	21	00010101	Negative acknowledge (tidak tampak)
SYN	0016	22	00010110	Synchronous idle (tidak tampak)
ETB	0017	23	00010111	End of transmission block (tidak tampak)
CAN	0018	24	00011000	Cancel (tidak tampak)
EM	0019	25	00011001	End of medium (tidak tampak)
SUB	001A	26	00011010	Substitute (tidak tampak)
ESC	001B	27	00011011	Escape (tidak tampak)
FS	001C	28	00011100	File separator
GS	001D	29	00011101	Group separator
RS	001E	30	00011110	Record separator
US	001F	31	00011111	Unit separator
SP	0020	32	00100000	Spasi
!	0021	33	00100001	Tanda seru (exclamation)
"	0022	34	00100010	Tanda kutip dua
#	0023	35	00100011	Tanda pagar (kres)
$	0024	36	00100100	Tanda mata uang dolar
%	0025	37	00100101	Tanda persen
&	0026	38	00100110	Karakter ampersand (&)
‘	0027	39	00100111	Karakter Apostrof
(	0028	40	00101000	Tanda kurung buka
)	0029	41	00101001	Tanda kurung tutup
*	002A	42	00101010	Karakter asterisk (bintang)
+	002B	43	00101011	Tanda tambah (plus)
,	002C	44	00101100	Karakter koma
-	002D	45	00101101	Karakter hyphen (strip)
.	002E	46	00101110	Tanda titik
/	002F	47	00101111	Garis miring (*slash*)
0	0030	48	00110000	Angka nol

1	0031	49	00110001	Angka satu
2	0032	50	00110010	Angka dua
3	0033	51	00110011	Angka tiga
4	0034	52	00110100	Angka empat
5	0035	53	00110101	Angka lima
6	0036	54	00110110	Angka enam
7	0037	55	00110111	Angka tujuh
8	0038	56	00111000	Angka delapan
9	0039	57	00111001	Angka sembilan
:	003A	58	00111010	Tanda titik dua
;	003B	59	00111011	Tanda titik koma
<	003C	60	00111100	Tanda lebih kecil
=	003D	61	00111101	Tanda sama dengan
>	003E	62	00111110	Tanda lebih besar
?	003F	63	00111111	Tanda tanya
@	0040	64	01000000	A keong (@)
A	0041	65	01000001	Huruf latin A kapital
B	0042	66	01000010	Huruf latin B kapital
C	0043	67	01000011	Huruf latin C kapital
D	0044	68	01000100	Huruf latin D kapital
E	0045	69	01000101	Huruf latin E kapital
F	0046	70	01000110	Huruf latin F kapital
G	0047	71	01000111	Huruf latin G kapital
H	0048	72	01001000	Huruf latin H kapital
I	0049	73	01001001	Huruf latin I kapital
J	004A	74	01001010	Huruf latin J kapital
K	004B	75	01001011	Huruf latin K kapital
L	004C	76	01001100	Huruf latin L kapital
M	004D	77	01001101	Huruf latin M kapital
N	004E	78	01001110	Huruf latin N kapital
O	004F	79	01001111	Huruf latin O kapital
P	0050	80	01010000	Huruf latin P kapital
Q	0051	81	01010001	Huruf latin Q kapital
R	0052	82	01010010	Huruf latin R kapital
S	0053	83	01010011	Huruf latin S kapital
T	0054	84	01010100	Huruf latin T kapital
U	0055	85	01010101	Huruf latin U kapital
V	0056	86	01010110	Huruf latin V kapital
W	0057	87	01010111	Huruf latin W kapital
X	0058	88	01011000	Huruf latin X kapital
Y	0059	89	01011001	Huruf latin Y kapital
Z	005A	90	01011010	Huruf latin Z kapital
[	005B	91	01011011	Kurung siku kiri

\	005C	92	01011100	Garis miring terbalik (*backslash*)
]	005D	93	01011101	Kurung sikur kanan
^	005E	94	01011110	Tanda pangkat
_	005F	95	01011111	Garis bawah (underscore)
`	0060	96	01100000	Tanda petik satu
a	0061	97	01100001	Huruf latin a kecil
b	0062	98	01100010	Huruf latin b kecil
c	0063	99	01100011	Huruf latin c kecil
d	0064	100	01100100	Huruf latin d kecil
e	0065	101	01100101	Huruf latin e kecil
f	0066	102	01100110	Huruf latin f kecil
g	0067	103	01100111	Huruf latin g kecil
h	0068	104	01101000	Huruf latin h kecil
i	0069	105	01101001	Huruf latin i kecil
j	006A	106	01101010	Huruf latin j kecil
k	006B	107	01101011	Huruf latin k kecil
l	006C	108	01101100	Huruf latin l kecil
m	006D	109	01101101	Huruf latin m kecil
n	006E	110	01101110	Huruf latin n kecil
o	006F	111	01101111	Huruf latin o kecil
p	0070	112	01110000	Huruf latin p kecil
q	0071	113	01110001	Huruf latin q kecil
r	0072	114	01110010	Huruf latin r kecil
s	0073	115	01110011	Huruf latin s kecil
t	0074	116	01110100	Huruf latin t kecil
u	0075	117	01110101	Huruf latin u kecil
v	0076	118	01110110	Huruf latin v kecil
w	0077	119	01110111	Huruf latin w kecil
x	0078	120	01111000	Huruf latin x kecil
y	0079	121	01111001	Huruf latin y kecil
z	007A	122	01111010	Huruf latin z kecil
{	007B	123	01111011	Kurung kurawal buka
¦	007C	124	01111100	Garis vertikal (pipa)
}	007D	125	01111101	Kurung kurawal tutup
~	007E	126	01111110	Karakter gelombang (tilde)
DEL	007F	127	01111111	Delete

2. Daftar Perintah Assembly untuk mesin intel x86

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.

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.