Laporan Praktikum Sistem Operasi
Kegiatan 1

Universitas Muhammadiyah Surakarta
28 September 2020

Disusun oleh :
Nama : Panji Lanang Samodra
NIM : L200190220
Kelas : F
Dosen Pengampu : Heru Setya Nugraha, ST., M.Kom

Tugas Kegiatan 1:

1. Apa yang dimaksud dengan kode ‘ASCII’ , buatlah table ASCII lengkap cukup kode ASCII yang standar tidak perlu extenced, tuliskan kode ASCII dalam format angka decimal, binary, dan hexadecimal serta karakter dan symbol 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 memahini program ‘boot.asm’ dan ‘kerpel.asm’.

JAWAB :

1. **ASCII** singkatan dari *American Standard Code for Information Interchange* atau Kode Standar Amerika untuk Pertukaran Informasi adalah standar pengkodean karakter untuk alat komunikasi. Kode ASCII mewakili teks dalam komputer, peralatan telekomunikasi, dan perangkat lainnya. Kebanyakan skema pengkodean karakter modern didasarkan pada ASCII, meskipun mereka mendukung banyak karakter tambahan. Kode Standar Amerika untuk Pertukaran Informasi atau sebuah standar internasional dalam pengkodean huruf dan simbol seperti Unicode dan Hex tetapi ASCII lebih bersifat universal.

Tabel ASCII

Dec	Bin	Hex	Char
32	010 0000	20	[space]
33	010 0001	21	!
34	010 0010	22	"
35	010 0011	23	#
36	010 0100	24	$
37	010 0101	25	%
38	010 0110	26	&
39	010 0111	27	'
40	010 1000	28	(
41	010 1001	29	)
42	010 1010	2A	*
43	010 1011	2B	+
44	010 1100	2C	,
45	010 1101	2D	-
46	010 1110	2E	.
47	010 1111	2F	/
48	011 0000	30	0
49	011 0001	31	1
50	011 0010	32	2
51	011 0011	33	3
52	011 0100	34	4
53	011 0101	35	5
54	011 0110	36	6
55	011 0111	37	7
56	011 1000	38	8
57	011 1001	39	9

Dec	Bin	Hex	Char
64	100 0000	40	@
65	100 0001	41	A
66	100 0010	42	B
67	100 0011	43	C
68	100 0100	44	D
69	100 0101	45	E
70	100 0110	46	F
71	100 0111	47	G
72	100 1000	48	H
73	100 1001	49	I
74	100 1010	4A	J
75	100 1011	4B	K
76	100 1100	4C	L
77	100 1101	4D	M
78	100 1110	4E	N
79	100 1111	4F	O
80	101 0000	50	P
81	101 0001	51	Q
82	101 0010	52	R
83	101 0011	53	S
84	101 0100	54	T
85	101 0101	55	U
86	101 0110	56	V
87	101 0111	57	W
88	101 1000	58	X
89	101 1001	59	Y
90	101 1010	5A	Z

Dec	Bin	Hex	Char
96	110 0000	60	`
97	110 0001	61	a
98	110 0010	62	b
99	110 0011	63	c
100	110 0100	64	d
101	110 0101	65	e
102	110 0110	66	f
103	110 0111	67	g
104	110 1000	68	h
105	110 1001	69	i
106	110 1010	6A	j
107	110 1011	6B	k
108	110 1100	6C	l
109	110 1101	6D	m
110	110 1110	6E	n
111	110 1111	6F	o
112	111 0000	70	p
113	111 0001	71	q
114	111 0010	72	r
115	111 0011	73	s
116	111 0100	74	t
117	111 0101	75	u
118	111 0110	76	v
119	111 0111	77	w
120	111 1000	78	x
121	111 1001	79	y
122	111	7A	z

	1010		
123	111 1011	7B	{
124	111 1100	7C	|
125	111 1101	7D	}
126	111 1110	7E	~
127	111 1111	7F	[del]

2. Daftar perintah bahasa Assembly untuk mesin intel keluarga x86 :

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 .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.

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.*